(1)

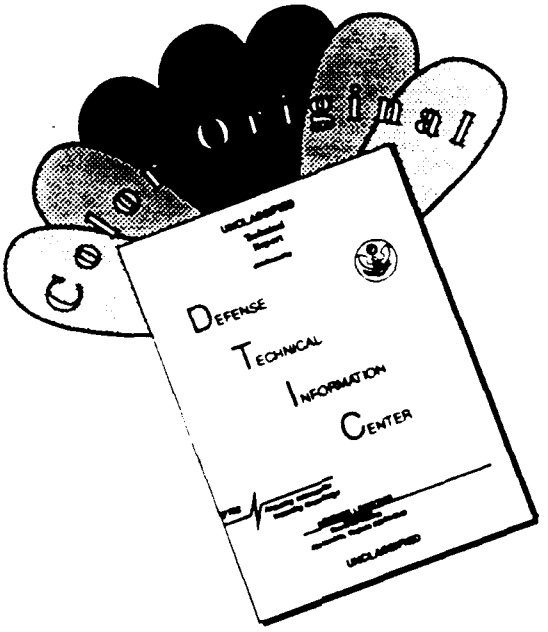
DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF COLOR PAGES WHICH DO NOT REPRODUCE LEGIBLY ON BLACK AND WHITE MICROFICHE.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1991	Master's Thesis

4. TITLE AND SUBTITLE
Separation of Cloud/No-Cloud Regions in Satellite Imagery Using a Variation of Hierarchical Clustering Analysis

5. FUNDING NUMBERS

6. AUTHOR(S)
Charles J. Martin, Jr., Captain, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology,
WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GSO/ENS/91D-12

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release;
distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This study investigated the usefulness of personal-computer-based software applying hierarchical clustering theory to try to separate cloud-covered regions from clear regions using Automated Picture Transmission imagery from the National Oceanographic and Atmospheric Administration's Television Infrared Observation Satellite. The algorithms were developed in Turbo Pascal, Version 6, and are part of the Training Software Image Processing program developed by a professor at the Air Force Institute of Technology. The goal of the project was to see if hierarchical clustering could provide better separation of cloud/no-cloud regions than an existing technique, histogram thresholding, while running on a personal computer.

Results of the research indicated that it was possible to use a centroid based clustering algorithm to separate cloud-covered regions from clear regions in APT imagery. Seed points were used to start the clustering process. The cloud seed point was chosen to be the brightest pixel in the clustering area. Typical results showed that the automated clustering approach provided results within 15 to 20 percent of those obtained from the histogram threshold method.

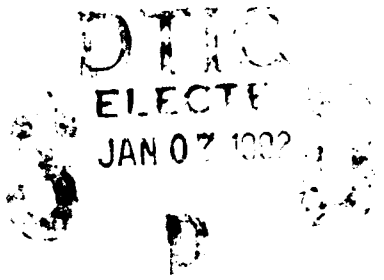
14. SUBJECT TERMS
Clustering, Image Dissection, Clouds, Classification
Cloud Identification, Image Processing, Weather Imagery

15. NUMBER OF PAGES
93

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

AFIT/GSO/ENS/91D-12



Separation Of Cloud/No-Cloud Regions In Satellite Imagery Using a Variation Of Hierarchical Clustering Analysis

THESIS

Charles J. Martin, Jr., Captain, USAF

AFIT/GSO/ENS/91D-12

THESIS APPROVAL

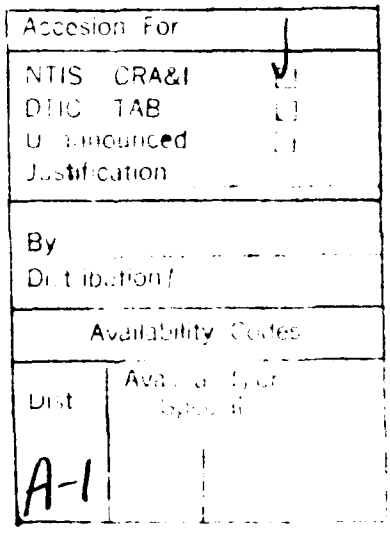
STUDENT: Charles J. Martin, Jr., Captain, USAF CLASS: GSO-91D

THESIS TITLE: Separation of Cloud/No-cloud Regions in Satellite Imagery Using a Variation of Hierarchical Clustering Analysis

DEFENSE DATE: 26 November 1991

GRADE:

COMMITTEE	NAME/DEPT	SIGNATURE
Advisor:	Major Thomas S. Kelso/ENS	Thomas Scan Kelso
Reader:	Lt Col Martin R. Stytz/ENC	Martin R Stytz



AFIT/GSO/ENS/91D-12

Separation Of Cloud/No-Cloud Regions In Satellite Imagery Using a Variation of Hierarchical Clustering Analysis

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Space Operations

Charles J. Martin, Jr., B.S.

Captain, USAF

December 1991

Preface

The purpose of this study was to develop a personal-computer-based program that utilized agglomerative clustering theory to try to separate weather features from non-weather features on Automated Picture Transmission (APT) imagery collected from the National Oceanographic and Atmospheric Administration's (NOAA) Television Infrared Observation Satellite (TIROS-N). Programming was done in Version 6.0 of the Turbo Pascal language, and a number of sample images were examined to explore the speed and accuracy of the program.

In researching and writing the algorithms contained within this report I have had a great deal of support from many individuals. I am deeply indebted to my advisor, Professor T. S. Kelso, who developed the program within which I have created the clustering algorithms. I would also like to thank my reader, Lt Col M. Stytz, who was very patient waiting for the first draft of this report. I would especially like to thank my fiancée, Susanne V. Lefebvre, whom I met while at the Air Force Institute of Technology. Her undying support got me through many tough situations. Finally, I would like to thank my parents for all that they have given me.

Charles J. Martin, Jr.

Table of Contents

List of Figures

List of Tables

Abstract

This study investigated the usefulness of personal-computer-based software applying hierarchical clustering theory to try to separate cloud-covered regions from clear regions. The weather data used was Automated Picture Transmission (APT) imagery collected from the Television Infrared Observation Satellite (TIROS-N) run by the National Oceanographic and Atmospheric Administration (NOAA). The imagery was collected and displayed using a small satellite receiver and personal computer set up by the Air Force Institute of Technology (AFIT) to study the effectiveness of receiving weather imagery on a relatively low-cost, and easy-to-transport system. The algorithms were developed within the TSIPS program, written by an instructor at AFIT, which had many of the routines necessary to support this project. In addition, the TSIPS program was written in Turbo Pascal specifically to run on a personal computer, making it easier to develop the clustering algorithms within it. The goal of the project was to see if hierarchical clustering could provide better separation of cloud/no-cloud regions than an existing technique, histogram thresholding, while running on a personal computer.

Results of the research indicated that it was possible to use a variation of the hierarchical clustering process to separate cloud-covered regions from clear regions. However, the results were not quite as good as those obtained from the histogram thresholding method. One advantage the automated clustering process has over the histogram process is that no user manipulation of a histogram is necessary in order to separate the clouds. Typical results showed that the automated clustering approach provided results within about 15 to 20 percent of those obtained from the histogram method. If the manual selection of seed points is used prior to running the clustering algorithms, the differences between the two methods virtually disappear.

Separation Of Cloud/No-Cloud Regions In Satellite Imagery

Using a Variation of Hierarchial Clustering Analysis

I. *Introduction*

Background

Sun Tzu, a famous Chinese tactician once wrote: "Know the ground, know the weather; your victory will then be total" (9:129). In order to "know the weather," it is necessary to be able to identify cloud formations and understand what they might mean. For centuries, ground-based observers were the only way to identify clouds, and their jobs were difficult due to their limited perspective of the clouds. After the invention of the weather satellite in the early 1960s, a whole new perspective of clouds was available that made identifying and predicting weather phenomena much easier. However, in the identification process it is assumed that the person looking at the sky, or [illegible]ery, is trained in the art of weather identification. In today's environment of spending reductions, the number of trained meteorologists is almost certainly going to be reduced both in the civilian and military workforce, which means that other methods of weather identification must be developed.

One alternate method of weather identification that has been under development for a few years uses computer analysis. Using a variety of techniques, computer algorithms will pick out weather features within an image, and provide a prediction as to what each feature is. Up to now, the majority of the work in this field has been done with mainframe computers, or workstations, because they alone had the processing power and storage capacity to run the algorithms. However, these systems have drawbacks, such as a lack of mobility, and a high price, which prevents their widespread use by organizations such as the armed forces which, by nature of its profession, needs mobility, but on a limited budget. This is where the personal computer can fit in. Since the early 1980s, when personal computers first appeared, their power and memory capacities have grown extensively. Computations that just a few years ago needed to be run on a mainframe computer can

today be run on a desktop system. Furthermore, advances in weather satellite technology have made weather imagery available to just about anyone, anywhere in the world, if they have the right equipment. Today, through the use of a simple receiver connected to a personal computer, it is possible to obtain satellite imagery from the Television Infrared Observation Satellite (TIROS-N) run by the National Oceanographic and Atmospheric Administration (NOAA) (2:Sec 4,1-2). The Air Force Institute of Technology (AFIT) operates such a system and can receive Automated Picture Transmission (APT) satellite imagery directly from the NOAA satellite and display it on a personal computer. Personal computers help solve the mobility and cost problems, but up to now there has not been any major software developed that can analyze weather features and be run on a personal computer.

There are a number of steps that must be accomplished to develop a program that can take APT imagery and identify the clouds within it. This project will deal with the first of these steps, the separation of cloud-covered areas from non-cloud-covered areas. Most of the weather identification algorithms already written to run on large computers use some type of single-image histogram manipulation to separate cloud-covered regions from clear regions. However, this type of analysis does not always completely separate cloud-covered regions from clear regions. Another technique, known as hierarchical cluster analysis, has shown to be very useful in other scientific fields, and may be applicable to this problem as well.

Research Objective

The objective of this research is two-fold. First, an exploration of the viability of using a hierarchical-clustering-based process to analyze NOAA APT satellite imagery and separate weather features from non-weather features will be conducted. Secondly, the programs implementing the clustering process will be written to run on a personal computer in order to determine if personal computers can be used for such a task. To achieve these objectives, the TSIPS package created by Professor T. S. Kelso will be used as the framework within which the clustering algorithms will be written. Ultimately, the work in this project could form the basis for a software package to analyze weather features on a personal computer.

Project Overview

The work done in an attempt to complete the project's objectives is discussed in the next four chapters. Chapter II provides background information in four key areas: NOAA APT imagery, TSIPS image processing software, current weather identification research with a specific focus on how cloud-covered regions are distinguished from clear regions, and hierarchical clustering theory. This background information helps build the foundation upon which the research in this project is based. Chapter III will examine how the hierarchical clustering theory was modified and turned into usable computer code, while Chapter IV discusses some examples of the finished algorithms at work. Chapter V will summarize the results of this research and provide some recommendations for future work. Finally, a user's manual for the computer algorithms, as well as the complete computer code for the clustering program can be found in the two appendices.

II. *Background Development and Literature Review*

The first step in accomplishing the objectives of this project is to develop some background information in four key areas. First, how the APT imagery is created will be examined. Second, the operation and hardware requirements of the TSIPS image processing program will be examined. Third, some current work in automated weather identification will be presented with a special focus put on how clouds are separated from everything else within an image. Finally, the basic theories of hierarchical clustering analysis will be presented, and to better understand how this process can be used, an application of its use to a real-world problem will be presented.

APT Imagery

The NOAA TIROS-N weather satellite has four different sensor packages onboard: Advanced Very High Resolution Radiometer (AVHRR), Operational Vertical Sounder (TOVS), Data Collection System (DCS), and the Space Environment Monitor (SEM) (2:2-1). The AVHRR sensor is used to image weather patterns in the earth's atmosphere and operates on four or five different channels, each of which monitors a specific band of wavelengths (2:Sec 2,1-3). The wavelength band and primary purpose of each channel is shown in Table 1.

Table 1. AVHRR Sensor Channels (Adapted from 2:2-3)

Sensor Channel	Wavelengths (micrometers)	Primary Use
1	.55 - .90	Daytime cloud and surface mapping
2	.725 - 1.1	Surface water delineation
3	3.55 - 3.93	Sea surface temperature, nighttime cloud mapping
4	10.5 - 11.5	Sea surface temperature, day/night cloud mapping
5	11.5 - 12.5	Sea surface temperature

The collected AVHRR imagery is transmitted to ground stations in two separate signals. The first signal, High-Resolution Picture Transmission (HRPT), is a direct readout of all channels of AVHRR data (2:2-1). The HRPT signal has a high data transmission rate (665.4 kilobits per second) due to the large amount of information that is being relayed (2:4-2). This high data rate requires that very sophisticated and expensive receivers be used to collect the data. The cost and complexity of HRPT receivers preclude many individuals and organizations from having access to the data, so a second data signal, Automated Picture Transmission (APT), is broadcast. To achieve slower data rates, the APT signal contains information from only two AVHRR channels, visible and infrared, and in each channel only every third line of the HRPT data is transmitted (10:134). The reduction in data allows the APT signal to broadcast at a rate of 8 kilobits per second which is accessible by less sophisticated receivers (2:4-2). The reduction in transmitted data also means a decrease in image resolution from 1.1 kilometers (HRPT) to 4 kilometers (APT) (2:2-1,10:134).

The APT signal starts off in a digital format onboard the satellite and is converted to an analog signal for transmission (7:38). The ground receiver digitally samples the analog signal 9600 times each second and assigns an integer value between 0 and 255 to each sample (6). The integer value corresponds to a specific gray-shade that can be used to display the sample on the computer monitor. The displayed image will be slightly degraded in resolution due to a smoothing process which results from the APT data being converted to an analog format, and then back to a digital format (7:39). However, the degradation is not enough to cause problems with the algorithms developed in this project, or with the identification of most weather phenomena due to their large-scale features.

Image Processing Software

The TSIPS image processing program was written by Professor T. S. Kelso, at the AFIT School of Engineering, for a class on the analysis of spatial and temporal modeling. The program is written in Turbo Pascal, Version 6.0, and can be used to carry out a variety of image analysis techniques upon satellite imagery including filtering, histogram manipulation, and contouring. The software requires an IBM-compatible computer system running the Microsoft™ Disk Operating System (MS-

DOS) with a Video Graphics Array (VGA) card and monitor. In order to get usable graphics images on the screen, the VGA adapter should be capable of displaying a minimum of 640 x 400 pixels in 256 possible colors. It is also recommended that the computer being used to run this software have at least 640 kilobytes of memory. In addition, for the clustering algorithms developed within this project to work efficiently, the computer system should have at least two megabytes of extended memory. A hard disk is also recommended for storage of APT images which are each about one megabyte in size.

Once started, TSIPS divides the monitor's screen into four equal-sized windows. Windows One through Three, which are numbered clockwise starting in the upper left, are used to display satellite imagery and perform operations on satellite imagery such as zooming or filtering. Window Four is used to display menus and supplementary information, such as histograms. Figure 1 shows what the main TSIPS screen looks like after having an APT image placed in Windows One and Two. The main TSIPS menu is visible in Window Four.

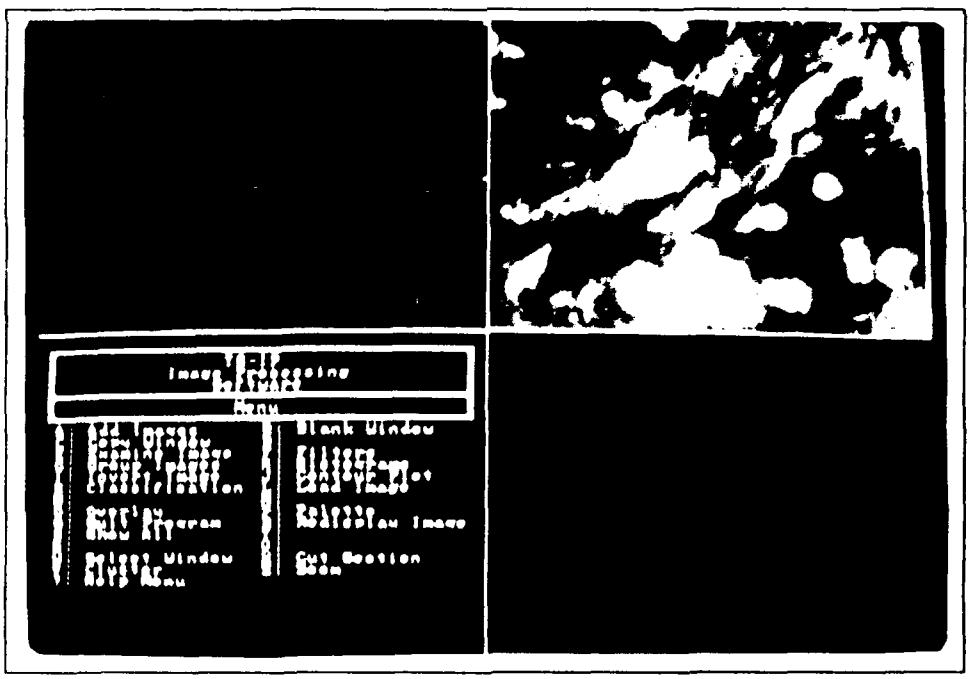


Figure 1. Main TSIPS Screen

One option within the TSIPS program that is used extensively in this project is the *GROUP IMAGES* option. This option allows the user to display the corresponding area from the visible and

infrared channels of the APT imagery in separate windows on the display. For instance, if a section of a visible APT image is being viewed in Window One, the user can select the *GROUP IMAGES* option and have the infrared image corresponding to the same section displayed in Window Two or Three. For the TSIPS program to accomplish this, preprocessing of the APT imagery is required to translate the visible and infrared images into a common coordinate frame. The screen shown in Figure 1 has a visible APT image in Window One, and its corresponding infrared image in Window Two. The color palette used to display the imagery is an option the user can select using the TSIPS program. Further information on the TSIPS package can be found in Appendix A, and copies of the software, along with full documentation, are available from Professor Kelso.

Weather Identification Using Computer Algorithms

In this section, a review of two articles that discuss the development of automated weather identification packages will be presented. The focus of these reviews will be on how the articles discussed separating the cloud-covered regions from everything else within an image. The first article to be reviewed discusses the feasibility of a project known as Short-range Expert Analysis and foreCAST, or SEACAST.

SEACAST. The primary purpose of this project is to provide aircraft-carrier-based meteorologists an artificial-intelligence-based computer program to help identify weather patterns (4:13). The satellite imagery being used in this project is received from the Geostationary Operational Environmental Satellite (GOES), which provides four-kilometer resolution visual imagery and eight-kilometer resolution infrared imagery (4:14). Predominately, the authors used the visible imagery as the initial indicator of cloud/no-cloud regions as it provides the best contrast between clou loud regions. In cases where this image is not available, mainly at night, the infrared image was used, but with less confidence. The article focused on the initial stages of the project where the authors chose to focus on the identification of stratus and stratocumulus clouds in the Pacific Ocean between California and Hawaii (4:13). This region frequently exhibits large amounts of both types of clouds.

The first step in the identification process is to create a histogram of the pixel values within an image. A histogram simply shows the number of pixels that exhibit each of the possible brightness values contained within an image. The researchers found that when trying to identify stratus or stratocumulus clouds over an ocean, the histogram of the visible image normally contained two dominant peaks. One of the peaks corresponds to the large number of pixels that have brightness values denoting them as ocean surface, while the other peak represents pixels with brightness values corresponding to clouds. An example of a histogram was not presented in the article, however, Figure 2 shows a histogram that would look a lot like the one described. The y-axis in Figure 2 is unlabeled, but represents the number of pixels.

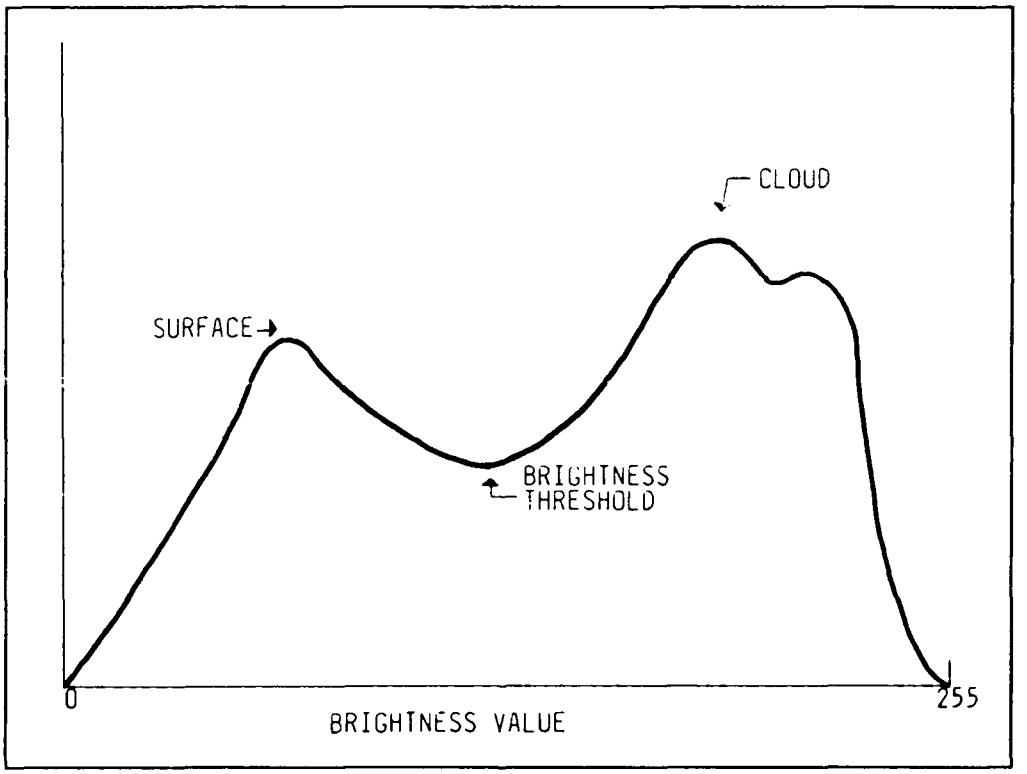


Figure 2. A Sample Histogram (Reprinted from 3:43)

To separate the clouds from the ocean surface, the authors simply set the minimum brightness threshold of the image at the minimum point between the two peaks on the histogram. Thus, all the pixels that had brightness values associated with the ocean surface were effectively turned off, and only the pixels corresponding mainly to clouds were left on. The next step in the identification

process involved using the infrared image to determine temperatures of different regions within the image. Only regions that had an infrared image corresponding to a temperature of 269 Kelvins or greater were used, as this was the temperature threshold that the authors determined could separate stratus/stratocumulus from other types of clouds (4:15). Finally, using a technique known as morphological filtering, the authors were able to get a good identification of areas within an image that corresponded to stratus or stratocumulus clouds. The authors concluded that their technique showed the promise of being able to be expanded to include the identification of other types of meteorological phenomena such as turbulence-formed clouds, and tropical systems (4:16).

No mention was made in the article about what type of computer system was being used to develop the algorithms, however, it seems that the techniques discussed would need to run on at a minimum, a fast personal computer, with a workstation being the most likely choice. The next article to be reviewed deals with the development of fast algorithms for the analysis of cloud data.

Ultrafast Algorithms for Cloud Data Analysis. In this article, the authors discuss some of the theoretical framework needed for developing software and hardware for the quick identification of clouds using satellite images. No specific examples are presented, however, as in the previously discussed project, the authors describe how a single-image histogram can be used to establish a threshold that can be used to separate clouds from other objects within an image. The drawback of this technique, however, is that the data from a single image does not always allow for the totally accurate identification of cloud-covered regions (3:38). The authors briefly address the use of multi-spectral data, but only to the extent of saying that if this type of data were being used, the algorithms and hardware would have to be modified to take into account multiple histograms. Furthermore, if the histograms created have more than one local minimum, unlike Figure 2, multiple thresholds will have to be established. One technique the authors mention to deal with multiple thresholds is to segment the original image by the ranges of brightness values between threshold points on the histogram. A relatively simple histogram, like Figure 2, exhibits only one local minimum, thus, only one threshold is needed. The image can be broken into two pieces by taking all the pixels with brightness values below the minimum and separating them from the pixels with

values above the minimum. This is exactly what was done in the SEACAST project. Another technique mentioned by the authors that could possibly be used for the separation and identification of clouds is called the nearest-neighbor method. This technique is more computationally intense than the histogram approach, but may yield better results. As will be shown in the next section, nearest neighbor is another name for the single-linkage hierarchical clustering process.

There are numerous other articles covering a wide range of attempts to identify clouds in satellite imagery. Many different types of digital image processing are being looked at for the identification of clouds, however, before the identification can be done, the clouds must be separated from everything else within the image. The majority of the projects reviewed use some type of histogram analysis for the separation process. The capability to do this type of analysis on satellite imagery is included in the program developed in this project. It will be used as a comparison to the results of the clustering routines to see which one provides better results.

Hierarchical Clustering Analysis

Hierarchical clustering analysis is one of many different forms of cluster analysis that researchers have been using for many years to attempt to find patterns within a set of data. The interest in clustering analysis has grown markedly over the years and it is being used in a number of scientific fields as a research tool. These fields include life sciences, medical sciences, social sciences, earth sciences, and even engineering sciences (1:5-6). For example, in the area of engineering sciences, clustering analysis has been used to help identify things such as radar signals and fingerprints (1:6).

Hierarchical clustering analysis can be divided into two areas; agglomerative and divisive (5:44). The agglomerative method involves starting with n objects or clusters and grouping them two at a time until all n objects are contained within a single cluster. The divisive approach is just the opposite. Starting with one large cluster, smaller clusters are broken out until only individual objects are left. Figure 3 shows a simple diagram that demonstrates the two clustering techniques.

In this project, the agglomerative process would involve starting with n clusters, each of which is a single pixel contained within an APT image, and grouping them together into larger and larger

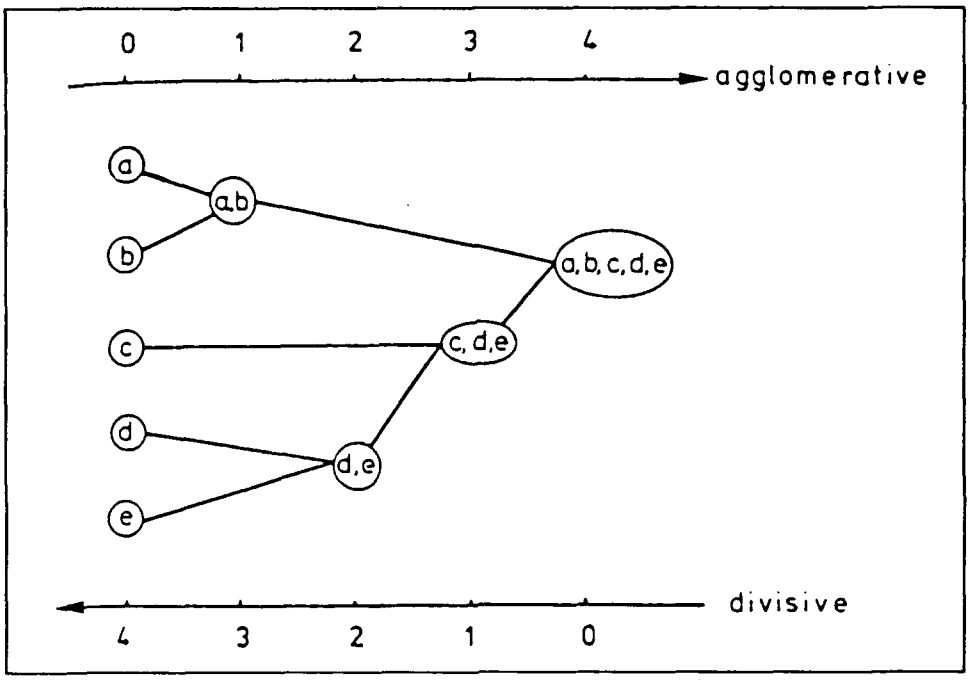


Figure 3. Hierarchical Clustering Techniques (Reprinted from 5:45)

clusters until the final cluster is the image on the screen. In contrast, the divisive process would begin with the entire image, and break it apart, pixel by pixel, until, after the last step, there were *n* clusters left each representing a single pixel. Divisive processes are not widely used due to the large amount of computations necessary to complete the process. In their books, both Kaufman and Anderberg discuss that most divisive methods, on just the first step, need to analyze $\{2^{n-1} - 1\}$ combinations, where n represents the number of individual objects (pixels), in order to divide the data set into two clusters. At each subsequent step, this number of calculations increases exponentially (5:253-254,1:155). In contrast, agglomerative methods, on the first step, consider $\{n(n-1)/2\}$ combinations, and for each subsequent step, the number of combinations grows quadratically with n (5:253). The result is that while both processes result in large amounts of computations, the agglomerative method is much more feasible. Because the processes are going to be developed on a personal computer, it makes sense to use the process that results in the lesser amount of calculations. Thus, it was decided that the clustering algorithms written would be designed around an agglomerative process.

In agglomerative hierarchical clustering analysis, the determination of which two objects, or pixels for this project, to cluster at any one time is frequently based upon a distance calculation performed with some form of the Minkowski distance formula shown in Equation 1 (5:13).

$$d(i,j) = \left(\sum_{k=1}^{p} |x_{ik}-x_{jk}|^{q}\right)^{1/q} \quad (1)$$

where

d(i, j) is the distance between two objects
q is any real number greater than or equal to one
p is the number of categories used to calculate the distance between objects
x_{ik} represents object i's value for the k^{th} category
x_{jk} represents object j's value for the k^{th} category

The distance between two objects, which is sometimes referred to as a similarity value, is calculated by taking the difference between both objects in a number of different categories and combining those differences into one equation. For example, if a two-dimensional plot was being used to determine the positions of objects, then the two categories of measurements could be the x and y position of the objects. Thus, one difference in Equation 1 would be the separation in the x coordinates of two objects, and a second difference would be the separation in y coordinates. This type of a distance calculation, used in combination with a q value of two, leads to a form of the Minkowski formula known as the Euclidean metric shown in Equation 2.

$$d(x,y) = \sqrt{(x_i-x_j)^2+(y_i-y_j)^2} \quad (2)$$

where

x_i, x_j represents the x position of the two objects
y_i, y_j represents the y position of the two objects

Another commonly used derivative of the Minkowski formula, where q is set to one, is known as the Manhattan metric. However, the Euclidean formula appears to be the most popular choice among researchers because it represents the true geometric distance between two objects (5:11).

These variations of distance formulas are the core of many hierarchical clustering processes, and the results obtained from these equations directly determine which objects are placed into which cluster.

One variety of agglomerative clustering that uses the distance formula is known as the single-linkage, or nearest-neighbor method (5:225-226). This process will attempt to group objects into clusters based upon the smallest distance between objects. For example, if a process begins with five objects, as in Figure 3, the first step would be to compute the distances between all possible combinations of objects. This means a total of ten distance values would be calculated. Next, the smallest of the ten values would be found, and the two objects associated with it would be grouped into one cluster. In Figure 3, this would be represented by the cluster (a,b). The next step would be to find the smallest distance value of the nine remaining, and put those two objects into a cluster. This step is represented by the cluster (d,e) in Figure 3. The third step would again involve finding the smallest distance value of the eight remaining, and grouping those two objects into a cluster. In this example, the single object, c, is grouped into the cluster already containing Objects d and e. This means that either the distance between Objects c and d, or c and e, was the smallest of the eight remaining. Now there are two clusters, (a,b) and (c,d,e), and the only step left is to combine them into one final cluster. The single-linkage method is unique in that when a cluster is formed, the objects making up that cluster must be stored so that the distance from every other object to every object within the cluster can be examined at each step. This means that on a computer, this type of process could require quite a bit of storage space depending on the original number of objects. One perceived drawback to this technique is that all it takes to group two clusters together is a single link between two objects. Thus, this technique often leads to a chaining effect where clusters that really are not closely associated are grouped together (5:226,1:138).

Another agglomerative clustering technique that uses distance calculations is known as the centroid method. Again, using Figure 3 as the example, the first step in this process is to compute the ten distance values. The next step would see the cluster (a,b) formed, but instead of keeping track of both objects in the cluster, an average value for the cluster in each category would be computed by taking the mean of the values of both objects. For example, if x and y positions were

being used to compute distances, then the newly formed cluster's x-value would be the average of Object a and Object b's x-value. The same applies to the cluster's y-value. Thus, instead of retaining two sets of distances for the cluster, only one set is needed. However, the centroid process requires that a number of the distance values be recalculated after each grouping because at each step, two objects or clusters are combined, and an average cluster is put in their place with a new value in each category. So, after the cluster (a,b) is formed, the distance calculations between Cluster (a,b) and Objects c, d, and e must be redone. Next, the cluster (d,e) is formed, and an average value in each category for this cluster is computed from the values of Objects d and e. There are now only three clusters to consider; Object c, Cluster (a,b), and Cluster (d,e). The three distances between these objects are calculated, and the smallest is chosen. In this example, that distance would be from Object c to Cluster (d,e). A new cluster is formed, (c,d,e), and a new set of average values is calculated for this cluster. The final step in this process would be to combine Clusters (a,b) and (c,d,e).

An advantage of the centroid method over the single-linkage method is in the amount of storage needed. In the single-linkage method, every object that existed when the process began must be kept track of throughout the entire clustering process. In addition, the cluster that each object is placed into must also be tracked. In contrast, the centroid method does not require that every object be stored, only that a running total of the values for each category, in each cluster, be retained. The average for the cluster is found by simply adding the newest object's values in each category to the cluster's running category totals, and then dividing by the total number of objects within the cluster. One advantage that the single-linkage method has over the centroid method is that with single linkage the distance values only need to be calculated one time, as opposed to the centroid method where every time a cluster is formed some of the distance values must be recomputed.

A third type of agglomerative clustering process that uses a distance formula is known as complete linkage, or the furthest-neighbor method. Complete linkage could be considered the opposite of the single-linkage method because the similarity between two objects or clusters is

defined as the largest distance between objects, or objects and clusters (5:226,1:138). At each step of the clustering process, the distance calculated between a cluster or object to all other clusters or objects within the data set is calculated just like in the other two processes. However, instead of looking for the smallest distance value, the largest possible distance value between two clusters or objects is found. Then, if the two entities were combined, it would be known that all the distances between objects in both the entities would be less than the maximum computed. For example, if two clusters were combined into one, the maximum distance found would represent the diameter of the smallest sphere which could enclose both of those clusters (1:138). Anderberg summarized the usefulness of complete linkage best when he wrote: "the interpretation of the clusters can be made only in terms of the relationships within individual clusters ... there is no particularly useful interpretation involving the differences between clusters" (1:139). Trying to compare strengths and weaknesses of the complete-linkage method to those of the single-linkage and centroid method is difficult because complete linkage is not used very often and represents a different approach to the clustering process. Suffice it to say, there probably are uses for this technique, but, for the purposes of this project, where the distances between objects or clusters is the primary concern, it does not apply.

After reviewing the three types of agglomerative clustering techniques, it was decided that the algorithms developed within this project would use the centroid method as their foundation. The decision to use the centroid process over the single-linkage process was based mainly on the fact that even though the single-linkage method could probably provide results faster than the centroid method, the storage space needed for the single-linkage process far exceeded that required for the centroid process. Storage space can be a limiting factor when dealing with personal computers, therefore, a method that uses the least amount of space is the most desirable. For this reason, the centroid process was chosen. If the centroid process provides good enough results, the additional expense of adding more storage space can be avoided.

The conversion of centroid clustering theory into usable, personal-computer-based algorithms involved modifying the method to use seed points. Seed points force the clustering process to begin

building clusters around specific objects, or pixels, and help eliminate the amount of calculations and storage needed to complete the process. This modification is discussed in Chapter III.

Normalization of Raw Category Data

Frequently, it is desirable to normalize the raw data used to compute the distance between two objects. This will factor out any possibility of one category of data having an undue influence on the overall distance. The choice of normalization is usually left up to the researcher. Raw data may inherently be weighted exactly how the researcher wants it, thus, the normalization process would not be necessary. However, for the most flexibility in assessing the contribution of different categories to the overall distances computed, normalization is the key. Once the normalization is completed, user-defined weights can be assigned to each category to reflect the amount of influence the user wants to place on a particular category. A common way of normalizing the raw input data is described very nicely by Kaufman in his book *Finding Groups in Data, An Introduction to Cluster Analysis*. The process he discusses is the basis for the normalization options available in this project's clustering algorithms.

To see the usefulness of normalization, assume that x and y positions are being used as the categories with which distances between objects are being computed. The distance formula would look just like Equation 2. Now if the objects had x and y values that exhibited the same range of values, it would not be necessary to normalize them. However, if the objects vary only slightly in their x positions, but vary greatly in the y direction, then the distance calculated with Equation 2 could be influenced much more by the y values since the difference in x positions would be small compared to the possible y differences. To keep this from happening, both the x and y values should be normalized so that their values vary over the same range.

The first step in normalizing the raw data is to find the mean for each category being used. In the example being discussed, it is necessary to find the mean of all the x and y positions for all the objects within the data set. The mean for each category is found by using Equation 3.

$$\bar{x}_i = \frac{1}{n}\sum_{j=1}^{n} x_{ij} \tag{3}$$

where

$\bar{x}_i$ is the mean of the i^{th} category
n is the number of objects in the data set
x_{ij} is the j^{th} object's value in the i^{th} category

Once the mean for both categories is found, a standard deviation, represented by Equation 4, or a mean absolute deviation, Equation 5, is calculated (5:8).

$$s_i = \sqrt{\frac{1}{n-1}\sum_{j=1}^{n} (x_{ij}-\bar{x}_i)^2} \tag{4}$$

where

$\bar{x}_i$ is the mean of the i^{th} category
s_i is the standard deviation of the i^{th} category
n is the number of objects in the data set
x_{ij} is the j^{th} object's value in the i^{th} category

$$a_i = \frac{1}{n}\sum_{j=1}^{n} |x_{ij}-\bar{x}_i| \tag{5}$$

where

$\bar{x}_i$ is the mean of the i^{th} category
a_i is the absolute mean deviation of the i^{th} category
n is the number of objects in the data set
x_{ij} is the j^{th} object's value in the i^{th} category

The mean absolute deviation can be useful because it is less susceptible to outliers in the raw data set than the standard deviation since the differences are not squared (5:8). Once the mean and the selected deviation are calculated, a normalized measurement is computed for both categories and put in place of the x and y values for each object. Equation 6 shows the typical way in which the

normalized value, sometimes referred to as the z-value, is calculated assuming that the standard deviation is being used (5:9).

$$z_{ij} = \frac{x_{ij} - \bar{x}_i}{s_i} \tag{6}$$

where

$\bar{x}_i$ is the mean value of the i^{th} category
z_{ij} is the z-value for the j^{th} object's i category value
x_{ij} is the i category value for the j^{th} object
s_i is the standard deviation of the i^{th} category

If the absolute mean deviation were to be used, simply use it in place of the standard deviation in Equation 6.

As stated before, normalization will remove any inherent weights within the raw data, leaving it up to the user to determine if any particular category should have more of an influence on the overall distance than any other category. To place added emphasis on a particular category, the distance equation can be modified to use weights. Equation 7 demonstrates what Equation 2 would look like with the addition of weight variables.

$$d(x,y) = \sqrt{w_x(x_i - x_j)^2 + w_y(y_i - y_j)^2} \tag{7}$$

where

w_x, w_y are the user defined weights for the x and y categories
x_i, x_j are the x positions of the objects
y_i, y_j are the y positions of the objects

Normalization can be a very useful tool, but it usually takes some trial and error to determine if it is a necessary step to complete in order to make sense out of a data set. Chapter III discusses how the normalization process was implemented in the clustering algorithms.

Application of the Agglomerative Clustering Method

To see how agglomerative clustering theory can be put to good use, one should read the article, *Infrared Analysis of Low Temperature Ashed Coal Ashes and Their Classification by Application of*

Clustering Theory, in the November issue of *Analytical Chemistry*. In the article, the authors describe how agglomerative clustering theory was used to group different types of coal into clusters based upon their absorbance at different wavelengths of energy. The authors began by collecting samples from 21 different types of coal that came from different locations around the world. They then ashed the samples through a process of grinding and oxidization. The different ashes were then compressed into pellets, and their infrared absorbance was studied with the use of an infrared spectrometer (8:2506). The authors documented absorbance characteristics at 40 different wavelengths in order to distinguish between coal samples.

Once data collection was complete, the authors normalized the absorbance values through the use of z-values, as discussed earlier. The standard deviation was used in the calculation of z-values, however, no reasoning was provided for this choice. Once the similarity values between all coal samples were computed, the single-linkage method was used to group the individual samples into clusters. Distances between objects or clusters were calculated using the Euclidean distance formula with 40 different categories. Figure 4 is a reproduction of the results obtained from the clustering algorithm and shows the same branching effect as seen in Figure 3. This is a typical way in which results from this type of clustering analysis are shown and is sometimes referred to as a hierarchical tree, or a dendrogram.

Notice how, by stepping one level down from the final grouping, two groups form; Australian coal, and everything else. Go down two more steps and notice how the data breaks out into four well separated groups: South African and Canadian coal; Belgium, American, and German coal; Australian coal; and the unknown coals. One can conclude much from looking at a diagram such as this. For example, Canadian coal is the most closely related to South African coal of all countries tested. In addition, Australian coal is very unique in that it isn't clustered until the last step which indicates that it varies significantly from all other types of coal sampled.

The authors suggest that more absorbance values could be used to arrive at more closely correlated clusters. This is a reasonable statement because as the number of categories used to calculate the distance between objects increases, the higher the certainty that two objects belong in

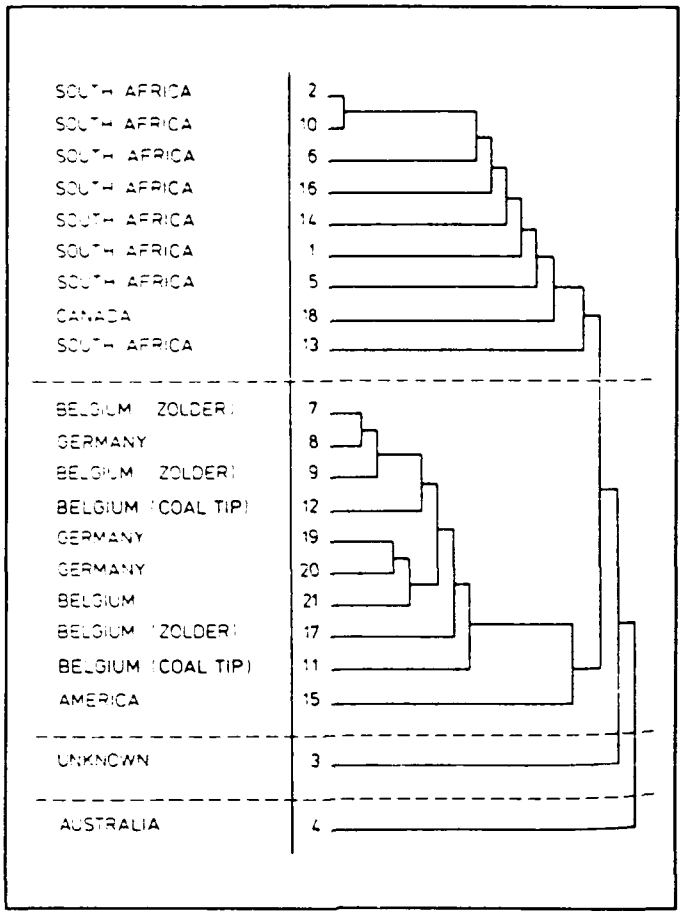


Figure 4. Classification of Coals (Reprinted from 8:2508)

the same cluster. For example, if only the x position was used to cluster two points on a two-dimensional plot, objects that have very close x-values would be clustered together even though they might have y-values that differed greatly. By including another measure, in this case y position, the certainty that the two points are grouped together properly will increase. This concept can be expanded to 40 dimensions in the case of the coal example, where each axis is the absorbance at a particular wavelength. As more wavelength measurements are added, the more certain the researcher can be that two objects are being grouped properly. No discussion was presented as to the amount of time required to complete this clustering analysis, nor was the type of computer system used mentioned. However, with only 21 objects, and 40 categories, this single-linkage example could probably be done easily on a personal computer.

Summary

Through use of APT imagery it is now possible to use a personal computer to view and manipulate satellite imagery. The TSIPS program written by Professor Kelso allows anyone to view and manipulate imagery using a variety of image analysis techniques. This program provides the ideal foundation upon which to build the clustering algorithms. The image manipulation and display functions have already been written, so attention can be focused on writing the actual clustering algorithms.

There has been much work in the area of weather identification using computer algorithms, however, most of it has been done on large computers. The process that is used most commonly for the separation of cloud-covered regions from clear regions is histogram thresholding. This process is quick, and relatively easy to implement, however, it is not always accurate. Another technique that could provide better results is an agglomerative clustering method. The centroid clustering method appears to be the best way to implement this process on a personal computer, based upon a balance between storage space and the amount of calculations necessary. By using seed points, the amount of storage and calculations can be reduced even further. Chapter III will discuss how the centroid clustering theory was modified to include seed points, and turned into the algorithms used in this project.

III. *Development of Computer Algorithms*

As mentioned in Chapter II, the TSIPS program was used as the framework within which the clustering algorithms were developed. TSIPS was written in Turbo Pascal, Version 6.0, and includes a variety of procedures that are used in this project. In addition, it was necessary to use the software package, *Huge Virtual Array and Numerical Analysis Toolbox*, developed by the Quinn-Curtis company which allows for the creation and manipulation of arrays in extended memory, or on a hard disk drive.

The computer system used in the development of these algorithms was based on an Intel™ 80386SX processor running at 20 megahertz. In addition, an 80387SX math coprocessor was present. Since the monitor used for the development of this software was only able to display 640 x 480 pixels, the largest section of an image that could be seen in any one window was 320 x 240. In order to keep the algorithms relatively simple, it was decided to use square areas of either 20 x 20 pixels, 50 x 50 pixels, 100 x 100 pixels, or 150 x 150 pixels as clustering examples. The average time for the program to completely cluster a 20 x 20 area of pixels was about two to three minutes. By going to a 50 x 50 area, the time increased to about 45 minutes for the clustering process to be completed. A 100 x 100 area was attempted, and the estimated time for its completion was about 40 hours. For the purpose of development and testing, the 20 x 20, and 50 x 50 areas were used most frequently.

Implementation of Centroid Method with Seed Points

In order to speed up the program, and to conserve additional memory, the centroid clustering process described in Chapter II was modified to include the use of seed points. Seed points are pixels chosen either by the algorithm, or the user, and are points where the clusters begin to build from.

As previously mentioned in Chapter II, the normal centroid clustering method involves computing all possible distances between all pixels and clusters and then searching them to find the

smallest value. Once this distance is found, the two objects associated with it are grouped into a new cluster. Average category values for the cluster are computed, then the distances between all the pixels and the new cluster are recalculated. This process continues until all the pixels are contained in one large cluster. The normal centroid process uses much less storage than other methods, but, by using seed points the amount of storage and the number of calculations needed can be decreased even further.

Once two seed pixels are chosen, only calculations from all the other pixels to the seed pixels need to be calculated. There is no need to calculate all the possible distances between non-seed pixels because the seed pixels tell the process where to start building the clusters. This results in significantly fewer calculations being conducted. For example, if a 3 x 3 pixel area were being examined, the total number of distances between all pixels that originally would have to be calculated is 36. By using two seed pixels, the number of distances drops to 14. The basic assumption this project is built on is rather simple. Clustering analysis is a way of looking for patterns within a data set. However, if a portion of a pattern is known, then clustering analysis can be modified to develop and separate that pattern from other objects or patterns within a data set. In this project, the user or the algorithm can pick out a cloud pixel and a non-cloud pixel rather easily given some assumptions that are discussed below. Thus, if the right categories are used when computing the distances between clusters and pixels, one cluster can build up the cloud-covered areas and the other cluster will contain whatever is left.

Overview of Algorithm Operation

Once the user starts the TSIPS program and loads the appropriate images into Windows One and Two, the clustering option can be chosen. When clustering is selected, the following list of options will appear in Window Four:

0 : Cluster Using Similarity Matrix and User-Defined Seed Points
1 : Cluster Using Similarity Matrix and Automatic Seed Points
2 : Show Ranges of Similarity Values from One User-Defined Seed Point
3 : Option 0 with Data Normalization
4 : Option 1 with Data Normalization
5 : Histogram of Clustering Area

Briefly, Option Zero allows the user to select the cloud and non-cloud seed pixels before beginning the clustering process. Option One invokes the automatic selection process. Option Two does not involve clustering pixels, rather, it allows the user to select a specific pixel then display pixels which fall within a variety of distance ranges from the chosen pixel. Options Three and Four are the same as Options Zero and One except that the raw category data is normalized. Finally, Option Five allows the user to display a histogram of the clustering area and perform a simple histogram thresholding process. These options are fully discussed in Appendix A.

Depending on the option chosen from the main clustering menu, the user can choose two seed pixels, or have the program do it automatically. In the automatic process, the computer will search all the pixels within the clustering area to find the brightest and dimmest visible pixel. These should correspond to a cloud-covered region and a clear region unless there is noise within the clustering area, or the area does not contain both cloud-covered and clear regions. It is assumed in this project that the images used will be absent from noise initially, or will have been preprocessed so that the noise has been removed. Furthermore, the algorithms were tested on areas that contained both cloudy and clear areas. If the area contains all clouds, then the automatic selection process will end up choosing one seed point on a lower, or dimmer, cloud type and the other on a higher, or brighter, cloud type. If no clouds are contained within the clustering area, the algorithms simply proceed in choosing whatever appears brightest and darkest in the image. Users can avoid some of the unknowns in this process by using the options containing manual input of seed pixels. The manual input process is discussed in the user's guide in Appendix A.

Once the two seed pixels are chosen, two arrays are created, one for the cloud seed pixel, and the other for the non-cloud pixel. These arrays are used to store the distance calculations from every pixel within the clustering area to each seed pixel. Equation 8 is the Euclidean distance formula used to calculate the distances between pixels, and between pixels and clusters.

$$d(pixel_1, pixel_2) = \sqrt{w_1(\Delta Vis)^2 + w_2(\Delta IR)^2 + w_3(\Delta XPos)^2 + w_4(\Delta YPos)^2} \qquad (8)$$

where

w_1, w_2, w_3, w_4 are the user-selected weights
ΔVis is the difference in visible brightness of pixels
ΔIR is the difference in infrared brightness of pixels
ΔXPos is the difference in x position of pixels
ΔYPos is the difference in y position of pixels

The program has options that will normalize the raw category data, if so desired by the user. If the data is normalized, the visible, infrared, x-position, and y-position information for each pixel is converted to a z-value using the technique discussed in Chapter II. The program utilizes the standard deviation when calculating the z-values as it is assumed that any noise, or outliers, will have been removed from the data prior to the clustering algorithms being run. When distances are computed, z-values are used in place of the raw category data in Equation 8. The use of normalization will cause a slight degradation in the speed of the program's execution because of the way in which the z-values are stored for each pixel. When the z-values are calculated, each category is placed in a separate two-dimensional array. Therefore, four arrays are created, each of which is the same size as the clustering area chosen. For example, using a 20 x 20 clustering area, the four arrays will be 20 x 20 in size, and each will contain one of the four categories of z-values. The program was written this way because at the time of its development the two-dimensional Quinn-Curtis arrays were thoroughly understood while the three-dimensional, and larger arrays, needed more research. When calculating the distances between pixels, each of the four arrays must be accessed to obtain the z-values used in Equation 8. This results in a slow down in program execution that is more obvious the larger the arrays become. The majority of the testing done in this project was with 20 x 20 and 50 x 50 clustering areas, and at these sizes, the normalization procedure hardly slows the clustering process at all. It should be possible to combine the data from the four arrays into one multi-dimensional array which would cut down on some of the time needed to access and search arrays. This has been left for a future version of the software package.

Once the two arrays corresponding to each seed pixel are filled with distance values, the next step is to search them for the smallest value. First, the array for the cloud seed pixel is searched for the smallest distance contained within it. Once found, this distance is then compared to all the

values in the non-cloud array. If there is no smaller distance value in the non-cloud array, then the pixel from the cloud array is added to Cluster One, which is the cluster that builds up the cloud-covered regions. If a smaller value was found in the non-cloud array, then the pixel corresponding to it is added to Cluster Two. Once a pixel is assigned to a specific cluster, a value of one or two is placed in its position in a tracking array. This tracking array is used to let the program know when a pixel has already been acquired into a cluster. Once a pixel is acquired, it is displayed in Window Three in the color denoting which cluster it is in. The program will no longer recalculate distance values for that pixel. This helps speed up the clustering process as more and more pixels are acquired. Once the smallest distance value is found, and the appropriate cluster incremented by one, the new cluster values are calculated by adding the visible, infrared, x, and y positions to the cluster's overall totals, and dividing by the new number of pixels in the cluster. This is a slight modification of the centroid method of clustering described in Chapter II.

The next step is to recompute the distance values in the array associated with the cluster that has changed. Using the distance formula shown in Equation 8, new values are computed for each pixel by taking the difference in the four categories between the pixel and the cluster's mean values. Once the array is updated, the process starts over again. The looping process continues until all the pixels are acquired into a cluster, or until the number of iterations requested by the user has been completed.

Assessment of Clustering Program's Effectiveness

A qualitative assessment of the effectiveness of the clustering options will be performed by visually comparing the results of the clustering programs with a typical histogram manipulation. The capability to set a lower brightness threshold in the cluster area is included in Option Five of the clustering main menu. With it, the user can decide what brightness value will be the cut-off for displaying pixels. For example, if the cluster area contains a lot of water and clouds, the histogram should show two distinct peaks. By looking at the histogram, the user can get an idea of what brightness value corresponds to the minimum between the two peaks. The user can then pick a pixel in the cluster area with this minimum brightness value, and all pixels with values below it will

be set to a brightness of zero, while all pixels equal to, or greater in brightness will be left alone. Thus, the cloud pixels will remain on, and the water pixels will be effectively turned off. The results of this manipulation are available for the user to compare the clustering results too. This comparison will form the basis of the qualitative assessment presented in Chapter IV. More information on how to use the histogram options is available in Appendix A.

Summary

By modifying the centroid clustering process to use seed points, the amount of storage space, and the number of computations needed to run the clustering program can be reduced significantly. This project operates on the idea that the researcher is starting with a data set that has known characteristics and patterns. Thus, it is not necessary to have the algorithms look for patterns, only build upon the ones designated by the researcher. The effectiveness of this modified clustering procedure will be examined in Chapter IV. A complete explanation of the six options available within the clustering program can be found in Appendix A.

IV. *Examples of Program Operation*

In this chapter, three examples of the clustering program's operation are presented, along with a discussion of the program's effectiveness in separating cloud-covered regions from clear regions. The three examples were chosen because they exhibited some extremes in the amounts of cloud-covered and clear area. Results of the clustering program are compared with results obtained from using the histogram threshold method mentioned in Chapter III. The comparisons are done qualitatively by visually looking at the two method's results, and estimating whether or not the clustering algorithms did a better job separating cloud regions from non-cloud regions.

Example One

For the first example, a 50 x 50 pixel area was chosen that had a mixture of clouds, both low and high, and ground features. The majority of the area was made up of terrain, with the main weather feature being a diagonal line of clouds that exhibited numerous brightness levels. The darkest region in the clustering area was a lake, and the brightest areas corresponded to tops of possible thunderstorm cells. The results of the four clustering methods used are summarized in Table 2.

Table 2. Example One Summary

Clustering Option	Category Weights	Results (% Clouds Separated)
Histogram Method	N/A	~ 99%
Option Zero	All set to one	~ 90%
Option Zero	Vis/IR = 1 X/Y = 0	~ 95%
Option One	All set to one	~ 70%
Option One	Vis/IR = 1 X/Y = 0	~ 75%
Option Three	All set to one	~ 80%
Option Three	Vis/IR = 1 X/Y = 0	~ 90%
Option Four	All set to one	~ 60%
Option Four	Vis/IR = 1 X/Y = 0	~ 70%

As seen in Figure 5, Window Four contains a histogram of the clustering area. The histogram exhibits one peak centered on a brightness value of about 75. This peak corresponds to ground features. After the peak, there is a steady drop off in numbers of pixels as the brightness values increase. Occasionally, certain brightness values exhibit individual spikes that correspond to different cloud layers. However, unlike Figure 2, there is not a definite minimum between two peaks that can be used as an automatic cutoff point. This means a trial-and-error process is needed in order to find the proper threshold. To set the histogram threshold, it is first necessary to have the histogram displayed in Window Four. Next, a pixel that has a brightness value corresponding to the desired threshold value is identified in Window One using the threshold option available in Option Five of the clustering program. This method is not exact, but after a few tries the majority of the cloud-covered areas were separated from the clear areas. The best results obtained from the histogram method are shown in Window Two of Figure 5.

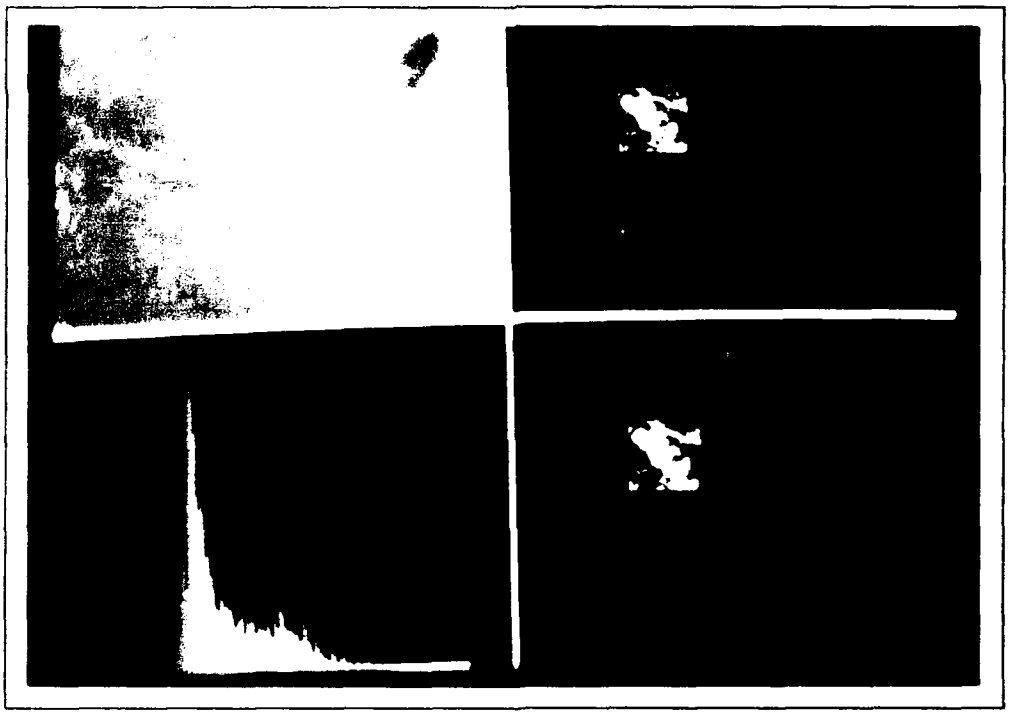


Figure 5. Example One (User Selected Seed Pixels)

The first clustering option run was Option One, where the program automatically selects the brightest and darkest pixels as the two seed points. Each of the four categories had a weight of one

assigned to them in the distance equation and data normalization was not used. The process took about 45 minutes to run and the results were acceptable. The main line of thunderstorms was easily identified in the cloud cluster, along with some of the middle-level clouds. However, most of the low clouds were not placed into the cloud cluster. A visual comparison of the best histogram results with those achieved with this option found that about 70 percent of the clouds were identified and clustered properly by the program.

Next, Option Zero, where the user selects the two seed points, was chosen. Rather than using the brightest pixel for the cloud seed point, a pixel corresponding to one of the lower clouds was chosen. The other seed point was placed within the dark lake region. The weights remained equal for all categories. The results were much better than those obtained with Option One. About 90 percent of the clouds were placed into the cloud cluster.

Option Zero was again chosen, only this time only the visible and infrared categories were used to calculate the distance values. This was accomplished by setting the x and y-category weights to zero in the distance equation. The results were even better then the previous trials, with about 95 percent of the clouds being clustered together. Figure 5 shows the clustering results, in Window Three, along with the histogram results in Window Two. About 95 percent of the clouds appeared to be clustered properly. There were only a few small discrepancies in comparison with the histogram produced results.

For the next trial, Option One was again selected, but only the visible and infrared categories were used. The results were better than those previously obtained with Option One, but not as good as those obtained with Option Zero. Around 75 percent of the clouds were placed in the cloud cluster. In Figure 6, the clustering results are shown in Window Three, along with the histogram results in Window Two. The next step was to examine the effects of using data normalization on the raw category data.

First, Option Three was chosen with all weights set to one. This option has the user select the seed points before the normalization process begins. A lower cloud pixel and a lake pixel were chosen as the two seed points. The results of this trial showed that about 80 percent of the clouds

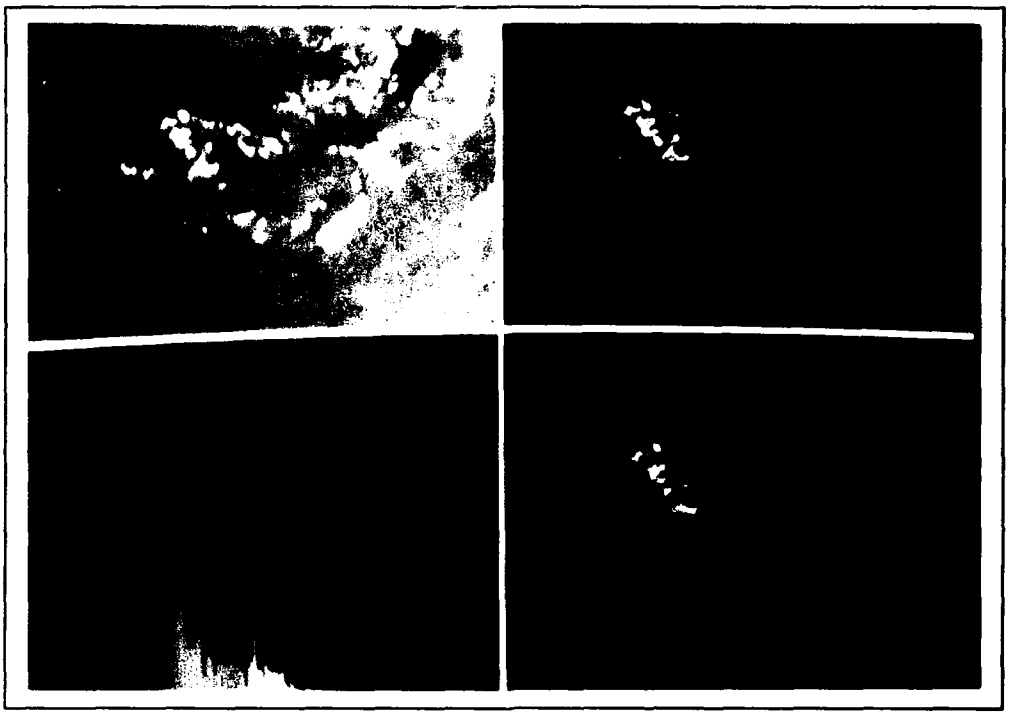


Figure 6. Example One (Automatic Selection of Seed Pixels)

where clustered properly. Again, Option Three was selected, only this time the x and y categories were not used. The results improved with an estimated 90 percent of the clouds clustered properly. It took about two minutes for the program to produce all of the z-values. After that was accomplished, the program ran just as fast as it did without normalization.

Next, Option Four was selected. This option implements the automatic selection of seed pixels. For the first trial, the weights were all set to one and the results were not very good. Only about 60 percent of the clouds were placed into the cloud cluster. Finally, only the visible and infrared categories were used, and the results improved to about 70 percent of the clouds being properly clustered. Overall, the results indicate that the best clustering option to use is Option Zero with only the visible and infrared categories contributing to the distance equation. The reason this option works the best seems to be based on a two things. First, by only using the visible and infrared categories, the positions of the pixels play no role in determining the distance values. Thus, a cloud pixel that is positioned a long way from the main cloud cluster can have a distance value comparable to a pixel that is right next to the cluster. This has the effect of increasing the likelihood that a cloud pixel will be placed into the cloud cluster. However, by using the x and y positions it is

possible to limit the growth of the clusters, thus allowing for more compact features to be attained. Second, by picking the cloud seed pixel to be part of a lower-level cloud, a minimum threshold is established, much like that of the histogram method, which means any pixel that has brightness values greater than the seed pixel will be placed into the cloud cluster.

Other combinations of weights were attempted to see if their results would fall within the ranges established by the trials shown in Table 2. No significant improvements were seen in any of the other trials. As one final test, Option Two was used to display different ranges of distance values. By placing the single seed point upon one of the brightest pixels, it was possible to display a number distance ranges that, when combined, gave results just as good as those obtained with the clustering options.

Example Two

For this example, another 50 x 50 area was chosen that contained one large cloud mass and a couple of small, low-cloud covered areas, all surrounded by water. The large cloud mass was roughly circular in shape and had two distinctive brightness levels. The smaller areas were predominately of one brightness value that was much dimmer then the main cloud mass. The results of the clustering options are summarized in Table 3.

Table 3. Example Two Summary

Clustering Option	Category Weights	Results (% Clouds Separated)
Histogram Method	N/A	~ 99%
Option Zero	All Set to One	~ 90%
Option Zero	Vis/IR = 1 X/Y = 0	~ 95%
Option One	All Set to One	~ 85%
Option One	Vis/IR = 1 X/Y = 0	~ 90%
Option Three	All Set to One	~ 90%
Option Three	Vis/IR = 1 X/Y = 0	~ 95%
Option Four	All Set to One	~ 80%
Option Four	Vis/IR = 1 X/Y = 0	~ 85%

As can be seen in Window Four of either Figure 7 or Figure 8, the histogram of this clustering area exhibited a distinctive peak centered on a brightness value of about 40, with a smaller peak centered on a brightness value of about 140. The larger peak corresponds to the water pixels, while the smaller peak is associated with the large cloud mass. It took a couple of attempts with the threshold method to separate the majority of the cloud pixels from the water pixels. The histogram threshold results are shown in Window Two of both figures.

All of the clustering options provided roughly the same results. Figure 7 shows the results of using Option Zero with only the visible and infrared categories contributing to the distance equation. This was the best of all the options and shows that the program did a good job of clustering most of the cloud pixels. Only small portions of the low-cloud areas were missed.

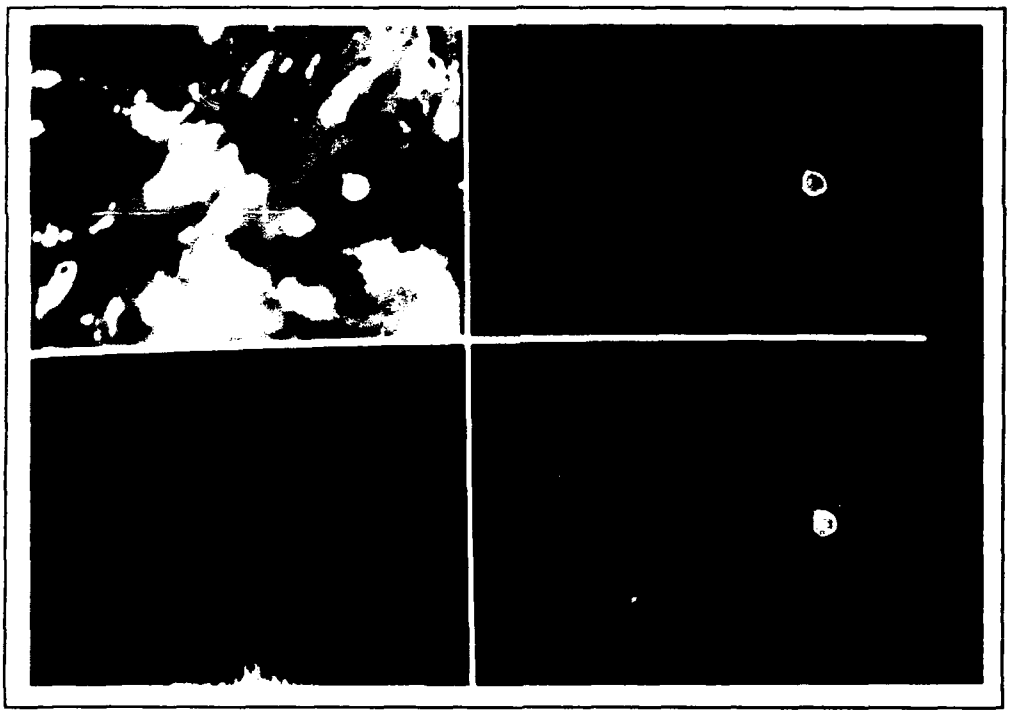


Figure 7. Example Two (User Selected Seed Pixels)

Figure 8 shows the results of using Option One with just the visible and infrared categories. This option provided the best results of all the automatic seed pixel options. The program clustered the large cloud mass, along with a portion of the lower cloud areas.

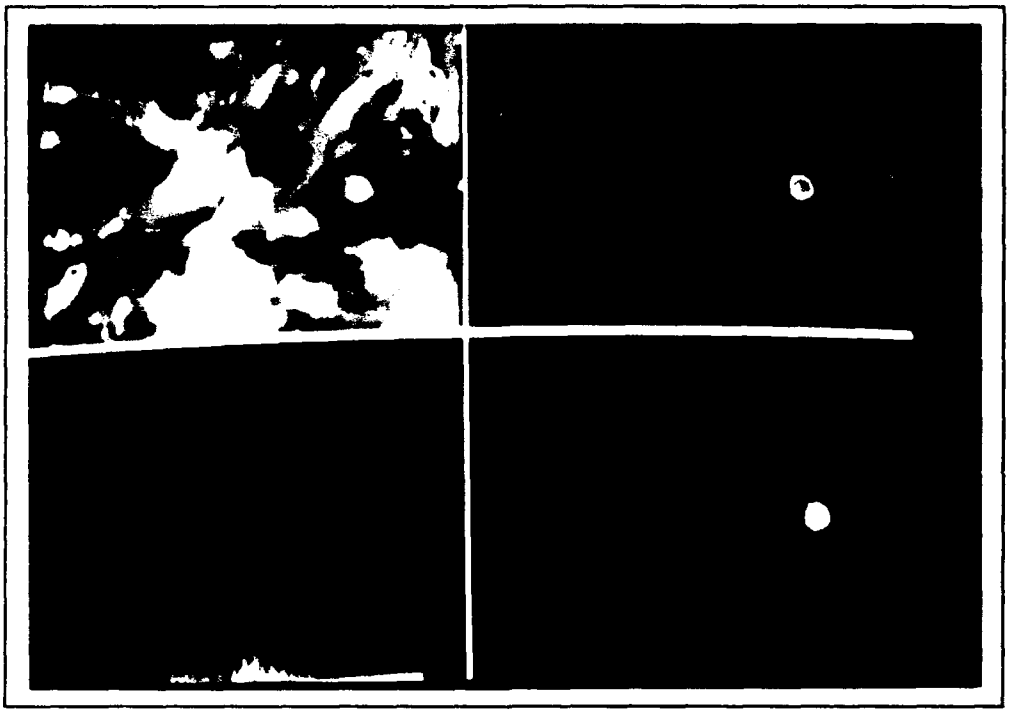


Figure 8. Example Two (Automatic Selection of Seed Pixels)

The simple cloud arrangement in this example, along with the distinctive brightness contrast between the cloud and non-cloud pixels, made practically any clustering option a viable one.

Example Three

The third example consisted of a 50 x 50 area composed mainly of clouds with just a small area of water visible. The cloud-to-cloud ratio was about opposite that in the first example. The results are summarized below.

Table 4. Example Three Summary

Clustering Option	Category Weights	Results (% Clouds Separated)
Histogram Method	N/A	~ 99%
Option Zero	All Set to One	~ 90%
Option Zero	Vis/IR = 1 X/Y = 0	~ 95%
Option One	All Set to One	~ 80%
Option One	Vis/IR = 1 X/Y = 0	~ 85%
Option Three	All Set to One	~ 80%
Option Three	Vis/IR = 1 X/Y = 0	~ 85%

Clustering Option	Category Weights	Results (% Clouds Separated)
Option Four	All Set to One	~ 70%
Option Four	Vis/IR = 1 X/Y = 0	~ 75%

As can be seen in Window Four of either Figure 9 or Figure 10, this example's histogram shows one definite peak around a brightness of 130, with smaller peaks at brightness values of about 30 and 80. It took multiple attempts using the histogram threshold method to pick the best threshold value.

The results of the clustering options again indicate that Option Zero is the best one to use. Once again, the best categories to use were only the visible and infrared, however, by including the x and y categories the results were not that much worse. The results of Option Zero using only the visible and infrared categories are shown in Figure 9.

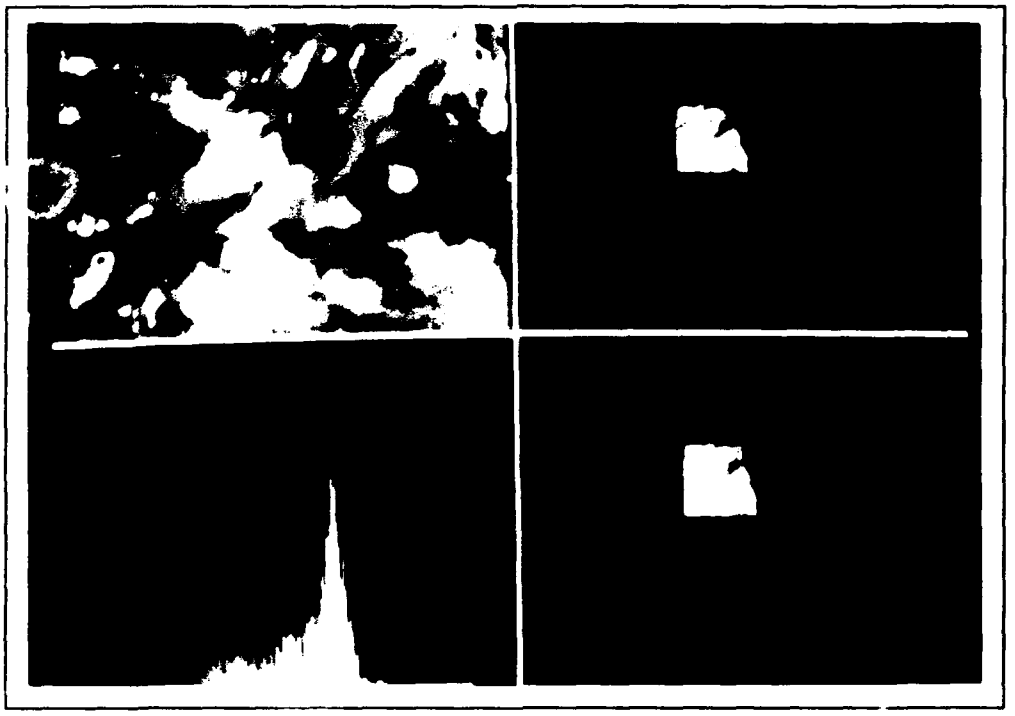


Figure 9. Example Three (User Selected Seed Pixels)

Figure 10 shows the results of Option One using only the visible and infrared categories to calculate the distance values.

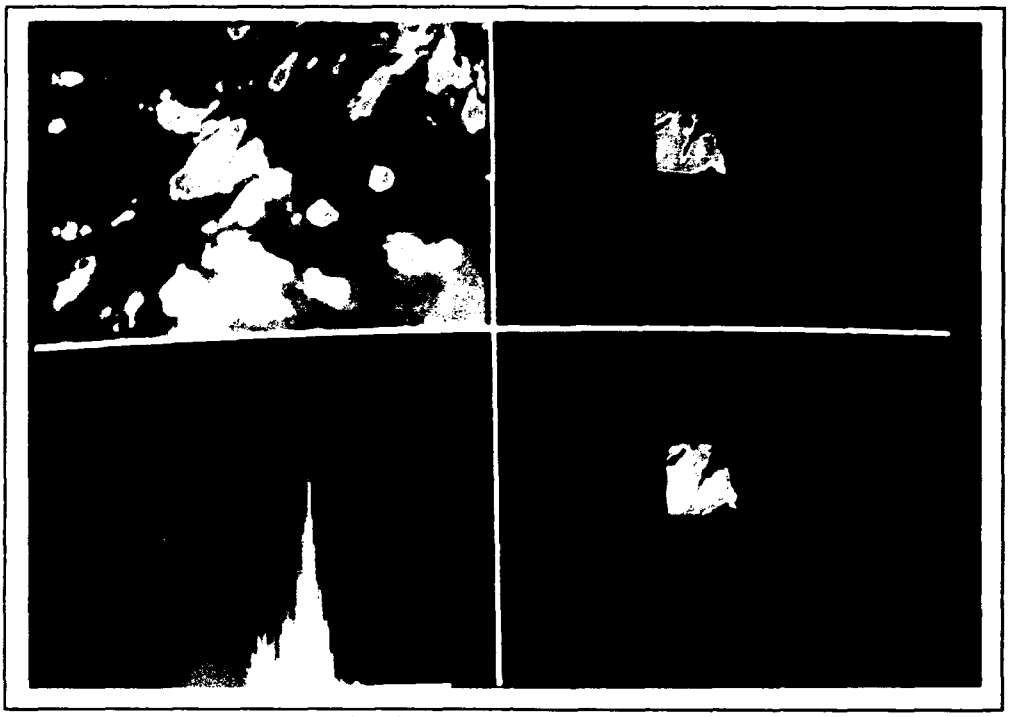


Figure 10. Example Three (Automatic Selection of Seed Pixels)

The options using automatic selection of seed pixels were unable to acquire a portion of the lowest cloud layer due to its closeness in visual and infrared values to the water pixels.

Summary

All the examples seem to have one thing in common, and that is they all indicate the best option to use is Option Zero with only the visible and infrared categories contributing to the distance equations. The x and y categories do not appear to add any additional help when trying to separate clouds that are scattered throughout the clustering area. The only time these categories could be useful is if the user is trying to isolate a particular area of clouds within an image. The x and y categories tend to impose a confining influence on how far away pixels can be and still be added to the cloud cluster. In addition, the use of data normalization did not appear to improve the clustering algorithms performance in any of the examples.

The histogram process was able to provide such good results for each example because the user took the time to find the brightness value corresponding to the lowest level of clouds and used it as the threshold. The key point here is that the histogram process worked because the user knew what

to look for. It is rather easy to tell if clouds exist in a satellite image just by looking at shapes and brightness values because large cloud masses rarely look like ground features. However, it is much more difficult to go pixel by pixel and identify which are cloud, and which are not. Thus, if someone trained to do the identification is not available, then an automatic process will have to be used.

If an automatic process is used, Option One with only the visible and infrared categories enabled is the best choice. For areas that had cloud regions scattered throughout, this process was able to separate about 75 to 90 percent of the clouds. The clouds that normally were not included were the lowest level clouds that had visible and infrared values very close to those of the terrain. The amount of low cloud included in the cloud cluster is dependent on the brightness difference between the cloud seed pixel and the lower-cloud pixels. If the difference is great, than the lower clouds will most likely be included in the non-cloud cluster. In each example, the amount of clouds identified was enough to provide an idea of the structure of the cloud mass. This means that identification of the cloud masses should be possible using some type of shape identification.

V. *Conclusions and Recommendations*

The goal of this project was to develop a personal-computer-based program that would use a variation of the centroid method of agglomerative hierarchical clustering to separate cloud regions from non-cloud regions in a satellite image. The results indicate that the process is possible, and can separate a large portion of the cloud features from the non-cloud features. However, in comparison to the histogram approach, the clustering approach did not provide quite as good results. However, the results obtained were good enough to determine the general shape and most of the substance of the clouds which means that it should be enough for a chance at identifying the clouds.

The choice of whether to use the histogram approach or the clustering approach depends on the needs of the program's user. If the user has the time to sit down and manipulate the histogram threshold to produce the best results, then the histogram approach is the best choice. However, if an automated process is desired, the histogram threshold technique may not work very well when there is no definite minimum in the histogram. For the three examples discussed in Chapter IV, only Example Two exhibited a histogram that had two well-defined peaks. However, it took a few attempts to find the best threshold value because the minimum between the peaks was not easy to locate. When there is no definite minimum, there is really no easy way a computer program can find the best threshold value unless someone is looking at the output and making the final decision. By using the program's automatic approach, where the brightest and dimmest pixels within the clustering area are chosen, the algorithms are able to separate about 75 to 90 percent of the clouds. Perhaps, rather than using the brightest pixel, it could be possible to obtain a better cloud seed pixel, (e.g., a lower cloud), by using a statistical analysis of the pixels within the clustering area. The examples show that if the cloud seed pixel is placed on a lower-cloud mass, the percentage of clouds properly clustered increases significantly. The automatic process was very good at identifying cloud areas that had a small range of brightness values, however, if the cloud cluster contained both very bright, and very dim pixels, the clustering process had a harder time identifying the lower clouds.

In terms of speed, the clustering algorithms are not nearly as fast as the histogram approach, however, whether or not speed is important depends upon the needs of the user. The speed of the program developed in this project depends on the processor being used, and the size of the clustering area. State of the art in personal computers is changing on a yearly basis, with speed being the feature that is increasing all the time. As far as the clustering area size is concerned, a 50 x 50 pixel area in an APT image covers an area of about 40,000 square kilometers. This is large enough to encompass practically any size military unit or formation. The intent is not to use the program to examine weather on a global scale, only to be able to let a field unit get an idea of what is over a projected target, or possibly in the area of the unit. The results obtained from the clustering algorithms should provide enough information for the identification of the major cloud masses.

A few areas could be looked at for further improving the algorithms developed in this project. First, implementing the single-linkage clustering method described in Chapter II might allow the clustering routines to provide better results. This technique looks at a lot more distance values when making a decision on which cluster to place a pixel in. Depending on the size of the cluster area being examined, it might be necessary to add more memory to the computer running the program. If the memory requirements become so large that only a hard disk drive provides enough storage for the arrays, then this method will be significantly slower than the centroid process due to the numerous disk reads needed. Second, an examination of alternate approaches to choosing the cloud seed pixel can be explored. Perhaps a method can be developed that uses the mean and standard deviation values of the histogram brightness values. For example, by selecting the brightness of the cloud seed pixel to be one or two standard deviations above the mean brightness level, a better choice of seed pixels may be made that would allow more of the lower cloud pixels to be put in the proper cluster. The examples discussed in Chapter IV indicate that choosing a lower cloud pixel as the seed point provides better results. Finally, it would be interesting to explore using more than two seed pixels in the clustering process. This would require a major modification to the software, but could allow more discrimination between cloud types or layers.

This project has shown that clustering can provide good results when trying to separate cloud-covered regions from clear regions in satellite imagery. The characteristic that makes this technique attractive is the lack of a need for a trained person to manipulate the image's histogram. The clustering technique, combined with a shape-identification algorithm, could provide an excellent tool for identifying clouds in satellite imagery. The next step is to explore the possibility of creating a personal-computer-based program to implement some type of shape-identification algorithm. If this is possible, then a personal-computer-based software package for identifying clouds in satellite imagery can become a reality.

Appendix A: User's Guide for the Clustering Algorithms

The first thing that must be done in order to run the clustering algorithms is to make sure that the configuration file for the TSIPS program is in the proper place and format. For TSIPS to run properly, make sure that it is located in a directory along with a configuration file entitled TS-IP.CFG. When the TSIPS program starts, it looks for the configuration file and reads it to determine where the image files are located as well as what palette to use upon starting. The configuration file can be created with a simple text editor. An example of a configuration file is shown in Figure 11.

```
2                   ; 640 x 480 at 256 colors
d:                  ; image_drive
\project            ; image_directory
d:                  ; work_drive
\project            ; work_directory
bgry.pal            ; default palette
```

Figure 11. Sample TSIPS Configuration File

The first line informs the TSIPS program which graphics resolution to use when displaying output on the computer monitor. There are five options represented by the numbers zero through four. Option Zero represents standard VGA with a resolution of 320 x 200 pixels in 256 colors. Option One has a resolution of 640 x 400 pixels in 256 colors. Option Two has a resolution of 640 x 480 pixels in 256 colors. Option Three has a 800 x 600 pixel resolution in 256 colors. Finally, Option Four has a resolution of 1024 x 768 pixels in 256 colors. Choose the option that corresponds to the graphics card and monitor being used to run the software.

The second and third lines tell TSIPS what disk drive and directory the satellite images are located in. The fourth and fifth lines are for locating the work drive and directories. The version of TSIPS used for this project did not use the work_drive or work_directory variables, however, they must be in the configuration file. The final line tells TSIPS what color palette to use upon starting. There are a number of different palettes available and they can be distinguished by the ".pal" suffix. Make sure that the palette files are located in the same directory as the TSIPS software.

Starting the Program

To start the program, enter TS-IP at the MS-DOS prompt and press ENTER. The first thing that will be seen is a title screen giving the name of the program along with the auth[illegible] and version date. Press any key to proceed to the main program screen. The screen will now be divided into four windows and the fourth window will have the main program menu displayed within it. The colors displayed will depend upon the startup palette the user placed in the TSIPS configuration file.

In order to use the clustering programs, a visible APT image must be in Window One, and the corresponding infrared image in Window Two. The program begins with Window One being the active window, so to load a visible image into this window, select the *LOAD* option from the main menu. A sub-menu will appear with a list of all the images available in the image directory specified in the configuration file.

The file names for each image give information on the date of the image, the NOAA satellite used, and what pass the image was taken on during the day. A complete description of the filename structure can be found in the TSIPS user's guide. Select a visible image file, which is distinguishable by a ".RSA" suffix, and press ENTER. The image will begin to load into Window One, going from top to bottom. If the image appears to be inverted, select the *INVERT IMAGE* option from the main menu. Reload the image and it will now be oriented in the proper manner. The inverted image comes from the fact that some of the weather satellites make North-to-South passes, while others make South-to-North passes over the earth's surface. If the image was taken during a South-to-North pass, the file will contain the information from the southern portion of the image first, thus, this is what will load first. Using the *INVERT IMAGE* option forces the images to load upside-down in the window, which means they will appear oriented the proper way. Once *INVERT IMAGE* is selected, all subsequent images will load from bottom to top in a window until *INVERT IMAGE* is chosen again.

Different portions of the visible image can be displayed in the window by using the arrow keys to move the image around. Once the chosen section of the visible image is displayed in Window One, select the *GROUP IMAGES* option from the main menu. A small box will appear in the

center of the screen with the number 1 highlighted within its upper left corner. This box is a representation of the active windows and is being displayed by the program because it is waiting for the user to select which window the infrared image should go into. Press the number 2 on the keyboard and then press ENTER. The second number will be highlighted briefly in the center box before disappearing. The infrared image corresponding to the visible image in Window One will now begin to load into Window Two. If the program stops executing, it will be because there is no corresponding infrared image. If this happens, start the program over and make sure when selecting the visible image that there is an infrared image associated with it. Infrared images will have the same file name as their corresponding visible image except for a ".RSB" suffix.

Now there should be a visible image in Window One and an infrared image in Window Two. It is time to start the clustering algorithms. Select *CLUSTERING* from the main menu and a second menu will appear asking the user what size of clustering area they wish to use. The version of the software used in this project has four options: 20 x 20 pixels, 50 x 50 pixels, 100 x 100 pixels, and 150 x 150 pixels. If the user wants to see the program in action but only take a small amount of time, select the 20 x 20 option. This option takes no more than two or three minutes to complete. The other options can take anywhere from about 45 minutes to over two days to complete, depending on the speed of the computer used. Once the area is selected, a box of the chosen size will appear in Window One. Move the box, using the arrow keys, over the area of the image that the user wishes to run the algorithms on. The distance the box moves for each press of an arrow key can be changed by using the PAGE UP and PAGE DOWN keys. For every press of the PAGE UP or PAGE DOWN key, the amount of the box movement for each press of the arrow keys will increase or decrease by roughly a factor of two. When the box is over the desired area of the image, press ENTER and that section of the visible and infrared image will be read into an array entitled *Parray*. The code for the reading process can be found on pages 58 and 59. For a 20 x 20 pixel area, the array will be 20 x 20 x 2 in size. The two values entered in the third dimension of the array are the visible and infrared values for each pixel.

After *Parray* is filled, another list of options will appear in Window Four. These options are as follows:

0 : Cluster Using Similarity Matrix and User-Defined Seed Points
1 : Cluster Using Similarity Matrix and Automatic Seed Points
2 : Show Ranges of Similarity Values From One User-Defined Seed Point
3 : Option 0 with Data Normalization
4 : Option 1 with Data Normalization
5 : Histogram of Clustering Area

If the user selects Option Zero, Two, or Three, it is up to the them to tell the computer where the seed pixels will be located within the clustering area. If Option One or Four is chosen, then the program will select the seed points automatically. As can be surmised from the option list, Options Zero, One, Three, and Four are all roughly the same, while Options Two and Five are different.

Memory Requirements

Before getting into how to use each option in the program, a special note about memory requirements is in order. The Quinn-Curtis routines used in this project will automatically create matrices and arrays in extended memory, if enough memory is available. If there isn't enough, the routines will then use the active hard disk drive as storage space. The computer being used to run these algorithms must have one or the other, otherwise the program will not work. Memory requirements for each of the clustering options vary depending on the option and the size of the clustering area chosen. For example, if a 20 x 20 area is chosen to be clustered and Option Zero or One is selected, then the following memory requirements are established. The distance arrays for each seed point will be 20 x 20 in size, with each value in the array being eight bytes in length. Thus, the required storage for these two arrays is 6400 bytes. In addition, the tracking array will be 20 x 20 in size and is filled with eight-byte values for an additional 3200 bytes of needed storage. The use of eight-byte values in these arrays is the default for the Quinn-Curtis routines. The option to change them is available to the user if a copy of the Quinn-Curtis software is obtained. The only array stored in the computers conventional memory is *Parray*. This array is used to store the visible and infrared values for each pixel in the clustering area. For a 20 x 20 area, *Parray* will by 20 x 20 x 2 in size with each of the two values being a byte in length. Thus, for a 20 x 20 area, the needed

storage space is 800 bytes. The largest clustering area possible within these routines is 150 x 150. This limit was set because Turbo Pascal imposes a limit of 64 kilobytes on any one array, so that if someone wanted to cluster an area bigger than about 180 x 180 pixels it would not fit into the memory Turbo Pascal allocates for it. To use larger arrays, the code could be modified to use the Quinn-Curtis routines to create the array in extended memory or on a hard disk drive.

The clustering options where normalization is conducted require an additional amount of space for each of the four z-value arrays. For a 20 x 20 area, each of these arrays will be 20 x 20 in size with each value in the array being eight bytes in length. Thus, all four arrays will require 12,800 bytes of storage space. The larger the clustering area chosen, the more space needed for each array.

Clustering Options

This section will describe each of the six options available in the clustering program. Option Five and Option Two are presented first because they stand apart from the other four options in that they provide a supporting role and do not actually cluster pixels. These two options provide additional information about the clustering area to the program user. The page references noted can be found in Appendix B.

Option Five. This option allows the user to see a histogram of the selected clustering area. The histogram will appear in Window Four and consists of 256 vertical bars, each in a different brightness value. The heights of the bars give the user an indication of the numbers of pixels that have the different brightness values. Once the histogram is displayed, Window Two will be cleared and a message will appear asking the user if he wishes to set a threshold for the clustering area. If yes is selected, a cross-hair will appear in Window One. The user can then use the arrow keys to move the cross-hair over a pixel with the desired threshold brightness value. The position of the cross-hair, in addition to the brightness value of the pixel it is over will be displayed in Window Four. The distance the cross-hair moves for each press of an arrow key can be increased or decreased by pushing the PAGE UP or PAGE DOWN keys. Once the user has selected the threshold value, the message in Window Two will disappear and the modified cluster area will take its place. The user can then compare the histogram threshold results with the clustering program

results shown in Window Three. When the user is finished with the histogram option, press the ESC key to return to the main TSIPS menu.

The histogram threshold process makes use of Window Two, which is also necessary for displaying the infrared APT image for the clustering algorithms to use. Once the visible and infrared values for the cluster area have been placed into the array *Parray,* it is not necessary to have the infrared image in Window Two. Thus, when this option replaces the infrared image, it is not going to affect the program. However, if any future clustering options are to be run, the infrared image must be placed back in Window Two before selecting the cluster option from the main TSIPS menu.

Option Two. This option allows the user to select a seed pixel within the clustering area, and have all the distance values for the other pixels displayed in specific, user selected ranges. To do this, press 2 on the keyboard and another menu will appear asking for the weights the user wishes to place on each of the four measures used in the distance calculation (see Equation 8). There are eleven preset options to choose from, or the user can select the letter *u* which allows them to enter their own choices for the weights. If they choose to do so, another display will appear in Window Four with the order and instructions on how to enter their chosen weights. This section of code can be found on pages 60 through 62. Once the weights have been chosen, the user will be asked to select the seed pixel they wish to use. A box will surround the clustering area in Window One, and a small cross-hair will appear in the center of the window. The cross-hair is moved with the arrow keys in the same manner described in Option Five. The PAGE UP and PAGE DOWN keys are available for increasing and decreasing the movement distance. A three-line display will appear in the upper-left corner of Window Four showing the cross-hair's x and y position, as well as the brightness of the pixel it is currently over. Place the cross-hair over the pixel within the clustering box chosen to be the seed point. Make sure that the pixel chosen is within the box or the program will not function properly. When the pixel is positioned properly, press ENTER. The code for the moving and selection of the seed point can be found on pages 51 and 52.

After the seed point is chosen, a list of options will appear in Window Four that represent the ranges of distance values the user can have displayed. The seed point chosen in Window One will be turned on in Window Three at this time. Select the range of distance values wanted by entering the appropriate number or letter on the keyboard. All the pixels within the clustering area that have a distance value from the seed pixel within the range chosen will be displayed in Window Three. The user can sequentially show more and more values in Window Three or has the option of changing palettes, clearing Window Three, or changing weights. When the user is finished displaying ranges of distance values, select ESC from keyboard and the main TSIPS menu will be displayed in Window Four. The code responsible for doing all of the calculations, as well as displaying of the distance values, can be found on pages 65 through 68. Comments in the code will explain what is going on at different points.

Options Zero and One. These two options differ only in the way the two seed points are selected within the algorithm. Option Zero allows the user to input his own seed pixels, while Option One has the computer automatically select the seed pixels. If Option Zero is selected, the user will first be prompted to select the desired weights for each of the four variables within the distance equation. The weight selection is identical to the method described in the previous section. Next, the user will be prompted to select the first seed pixel. When the cross-hair is over the desired seed pixel, press ENTER and then move the cross-hair to the second seed pixel. When properly positioned, again press ENTER. The code for the selection of seed pixels can be found on pages 51 and 52.

If Option One is selected, the program will look for the dimmest and brightest pixel within the image and use them as seed points. If more than one pixel is the dimmest or brightest, the first one encountered by the algorithm will be used. Once the two seed pixels are selected, the program will display them in Window Three and create the necessary distance and tracking arrays. The arrays are created either within extended memory or on the hard disk drive, depending on what is available, by using the *HMatDef* procedure available in the Quinn-Curtis software package. Each array is referenced by a pointer variable, so that when a read or a write is to be done to an array,

the programmer must simply specify the appropriate pointer in the read or write statement. The two arrays of distance values have the pointers *simptr*, and *simptr2* as their reference, while the tracking array has the pointer *ptr2*.

Once the arrays are created, the program will fill them with values by calculating the distance from each pixel to each seed point. Once both arrays are filled, the average values in the four categories for each cluster are set equal to the two seed pixels values (see page 70). The user is now prompted by a menu in Window Four to select the number of iterations the program is to complete before stopping. The number selected will be the total number of pixels added to the two clusters during the run of the program. If a 20 x 20 area is selected and the user wants all 400 pixels to be placed into clusters, then select 398 iterations. The two seed pixels start out in clusters, therefore, only 398 iterations need to be completed to place all pixels in a cluster. After the number of iterations is chosen, the program will begin to run through a looping sequence where the smallest distance value is found and then the pixel corresponding to it is placed with the appropriate cluster. After the pixel is added, the cluster's mean category values are updated, and the appropriate distance array is recalculated. Finally, a value of one or two is placed in the tracking array to keep track of which pixel was acquired and into which cluster it was placed. This loop will continue for the selected number of iterations, and each time the loop is completed, the iteration number will be updated in Window Four to show the user that the program is working. The complete code for these options can be found on pages 69 through 74.

Options Three and Four. These two options are almost the same as Options Zero and One except for an additional process that is added between the point where the user selects the seed pixels and the looping process begins. After the seed pixels have been chosen, either manually or automatically, the program will normalize the raw data in each of the four categories by computing *z*-values. To do this, the average value for each of the four categories is computed. For example, all 400 visible values for a 20 x 20 clustering area are added together and divided by 400 to compute a mean. Then a standard deviation is calculated using a formula like Equation 4. After the mean and standard deviation are computed, each pixel has its raw category values replaced with a *z*-value

computed from a formula like Equation 6. This process is done for the visible, infrared, x, and y values for each pixel. Each z-value has a specific array created for it, so when the normalization is complete, there are four 20 x 20 arrays filled with either visible z-values, infrared z-values, x-position z-values, or y-position z-values. Once the normalization is completed, the two distance arrays are then filled, only this time z-values are used in place of the raw data in Equation 8. When the arrays have been filled, the user is prompted to select the number of iterations the program is to complete. Then, both arrays are searched for the smallest value and the looping process described in the previous section is carried out. The code that is responsible for the computations described here can be found on pages 74 through 82.

Shut-Down of the Program

Once the selected clustering option has finished, the clustering program will free any memory associated with the arrays used in the process, remove any clustering boxes from Windows One and Two, and redisplay the main TSIPS menu in Window Four. The results of whatever clustering option selected remain in Window Three so the user can use other TSIPS options on the results, if so desired. At many points throughout each clustering option, the user has the ability to terminate the program's operation by selecting the ESC key. When this option is available, it will appear on a menu in Window Four. The code for the shut-down and housekeeping of the clustering algorithms is on pages 82 through 84.

Appendix B. Listing of Turbo Pascal Code

The following is the complete Turbo Pascal code for the clustering algorithms located within the TSIPS software. Comments are included to help the reader follow what is going on.

```
(* The following procedure displays the size of the cluster box the  *)
(* user requests from the main Cluster menu in Window One.  The box  *)
(* is displayed using the SetWriteMode (XORPUT) command.  This       *)
(* procedure is a modification of the Mark_Cut_Box procedure located *)
(* in the TSIPS program.                                             *)

Procedure Mark_Cluster_Box(sx, sy : integer;
                               sz : byte);
VAR

   sz2 : byte;

begin
     sz2 := sz div 2;
     SetWindow(active);
     SetWriteMode(XORPUT);
     Rectangle(sx-sz2,sy-sz2,sx+sz2-1,sy+sz2-1);
end; {Procedure Mark_Cluster_Box}


(* This procedure allows the user to move the cluster box around     *)
(* Window One so that the user can select the clustering area.  This *)
(* procedure is a modification of the Set_Cut_Block procedure        *)
(* located in the TSIPS program.                                     *)

Procedure Select_Cluster_Box(sz : byte;
                             VAR choice : char);
LABEL

   interrupt;

VAR

   sz2 : byte;
   lx, ly : integer;

begin
   sz2 := sz div 2;
   lx := -1;
   ly := -1;
   choice := ReadKey;
   repeat
         if choice = ESC then
            Goto interrupt;
```

```
         if (lx <> xx[active]) or (ly <> xy[active]) then
            Mark_Cluster_Box(xx[active],xy[active],sz);
         lx := xx[active];
         ly := xy[active];
         choice := Readkey;
         if choice = #00 then
            begin
            choice := ReadKey;
            if choice in [Up,Dn,Rt,Lt] then
               begin
               case choice of
                  Up : xy[active] := IMax(sz2,xy[active]-step);
                  Dn : xy[active] := IMin(xy[active]+step, cy-sz2);
                  Rt : xx[active] := IMin(xx[active]+step,cx-sz2);
                  Lt : xx[active] := IMax(sz2,xx[active]-step);
                  end; {case}
               if (lx <> xx[active]) or (ly <> xy[active]) then
                  Mark_Cluster_Box(lx,ly,sz);
               end {if}
            else if choice in [PgUp,PgDn] then
               case choice of
                  PgUp  : step := IMin(64,step shl 1);
                  PgDn  : step := IMax(1,step shr 1);
                  end; {case}
            end; {if}
      until choice in [^M,ESC];
      interrupt:

end; {Procedure Select_Cluster_Box}


(* This procedure allows the user to select specific pixels which    *)
(* are used as seed points for the clustering algorithms.  For the   *)
(* program to work properly, the pixels chosen as seed points MUST   *)
(* be within the clustering area.  This procedure is a modification  *)
(* of the Examine procedure located in the TSIPS program.            *)

Procedure Set_Cluster_Point(xs,ys : integer;
                            arg : byte;
                        VAR xcoord,ycoord : integer);

VAR

   choice          : char;
   win1,win2,pixel : byte;
   x,y,z           : string[3];

Procedure Show_Coordinates;
  begin
    SetWindow(4);
    Str(ex[active]:3,x);
    Str(ey[active]:3,y);
    Str(pixel:3,z);
    SetColor(0);
```

```
    Left_Text(1,'     ');
    Left_Text(2,'     ');
    Left_Text(3,'     ');
    SetColor(255);
    Left_Text(1,'X = '+x);
    Left_Text(2,'Y = '+y);
    Left_Text(3,'Z = '+z);
  end; {Procedure Show_Coordinates}

begin
    SetWindow(1);
    SetWriteMode(XORPut);
    Rectangle(xs,ys,xs+arg-1,ys+arg-1);
    SaveWorkSection(1,0,0,8*lw,5*lh);
    SetFillStyle(SolidFill,0);
    Bar(0,0,8*lw,5*lh);
    SetWindow(active);
    pixel := GetPixel(ex[active],ey[active]);
    Mark_Point(ex[active],ey[active],3,pixel);
    Show_Coordinates;
    repeat
    choice := Upcase(ReadKey);
    if choice = #00 then
      begin
      choice := ReadKey;
      if choice in [Up,Dn,Rt,Lt] then
        begin
        Unmark_Point(ex[active],ey[active],3);
        case choice of
          Up : ey[active] := IMax(0,ey[active]-step);
          Dn : ey[active] := IMin(ey[active]+step,cy-1);
          Lt : ex[active] := IMax(0,ex[active]-step);
          Rt : ex[active] := IMin(ex[active]+step,cx-1);
          end; {case}
        pixel := GetPixel(ex[active],ey[active]);
        Mark_Point(ex[active],ey[active],3,pixel);
        Show_Coordinates;
        end {if}
      else if choice in [PgUp,PgDn] then
        case choice of
          PgUp : step := IMin(64,step shl 1);
          PgDn : step := IMax(1,step shr 1);
          end; {case}
      end;{else}
  until choice = CR;
  xcoord := ex[active];
  ycoord := ey[active];
  Unmark_Point(ex[active],ey[active],3);
  RestoreWorkSection(1,0,0,8*lw,5*lh);
  SetWindow(1);
end; {Procedure Set_Cluster_Point}
```

```
(* This procedure moves the cursor on the text screen down into the  *)
(* area of Window Four so that when an option is chosen that has the *)
(* user input values with the keyboard, the values will not          *)
(* overwrite the image in Window One.  The user entered values are   *)
(* visible in Window Four if the active palette has a bright enough  *)
(* color at its low end.                                             *)

Procedure MoveCursor;

VAR

  MyRegs : Registers;

begin
  MyRegs.AH := $2;
  MyRegs.DH := 20;
  MyRegs.DL := 0;
  MyRegs.BH := 0;
  Intr($10,MyRegs),
end; {Procedure MoveCursor}


(* This procedure moves the cursor back up to the top of the text    *)
(* screen.                                                           *)

Procedure MoveCursor_Top;

VAR

  MyRegs : Registers;

begin
  MyRegs.AH := $2;
  MyRegs.DH := 0;
  MyRegs.DL := 0;
  MyRegs.BH := 0;
  Intr($10,MyRegs);
end; {Procedure MoveCursor_Top}


(* This procedure allows the user to see a histogram of the cluster  *)
(* area.  It also allows the user to set a minumum threshold and     *)
(* have all pixels below this threshold effectively turned off.  The *)
(* histogram is displayed in Window Four, and the modified cluster   *)
(* area in Window Two.  This procedure is a modification of the      *)
(* Histogram procedure located in the TSIPS program.                 *)

Procedure Cluster_Histogram(xs,ys : integer;
                            arg : byte);
LABEL

    interrupt1,interrupt2;
```

```
VAR

    k          : byte;
    i,j        : integer;
    value      : word;
    Multiplier: integer;
    pix        : byte;
    x,y,z      : string[3];
    pixel      : byte;
    choice     : char;

(* This small procedure is copied over from the Set_Cluster_Point     *)
(* procedure.  It has been slightly modified.                         *)

Procedure Show_Coordinates;
    begin
    SetWindow(4);
    Str(ex[active]:3,x);
    Str(ey[active]:3,y);
    Str(pixel:3,z);
    SetColor(0);
    Left_Text(1,'     ');
    Left_Text(2,'     ');
    Left_Text(3,'     ');
    SetColor(255);
    Left_Text(1,'X = '+x);
    Left_Text(2,'Y = '+y);
    Left_Text(3,'Z = '+z);
end; {Procedure Show_Coordinates}

(* Main histogram routine begins here.                                *)

begin
  hmin := 255;
  hmax := 0;
  for k := 0 to 255 do
    histogram[k] := 0;
  SetWindow(1);
  for i := 0 to arg-1 do
    for j := 0 to arg-1 do
      begin
      value := GetPixel(xs+j,ys+i);
      if value < hmin then hmin := value;
      if value > hmax then hmax := value;
      histogram[value] := histogram[value] + 1;
      end; {for j}
  SetWriteMode(XORPut);
  Rectangle(xs,ys,xs+arg,ys+arg);
  SetWindow(4);
  ClearViewPort;

(* These multipliers are used to scale the heights of the bars        *)
(* representing the number of pixels with a specific brightness value.*)
```

```
  if (arg = 20) then
    Multiplier := 3;
  if (arg = 50) then
    Multiplier := 2;
  if (arg = 100) then
    Multiplier := 1;
  if (arg = 150) then
    Multiplier := 1;
  for k := 0 to 255 do
    begin
    SetColor(k);
    Line(start+k,cy-01,start+k,cy-(histogram[k] * Multiplier)-01);
    end; {for k}

(* This information appears in Window Two and will clear away        *)
(* anything already displayed in the window.                         *)

  SetWindow(2);
  ClearViewPort;
  Center_Text(2,'Do you wish to set a threshold');
  Center_Text(3,'for the cluster area?  (y/n)  ');
interrupt2:
  Choice := Readkey;
  if (Choice =#110) then
    begin
      SetWindow(1);
      Rectangle(xs,ys,xs+arg,ys+arg);
      Goto interrupt1;
    end;

(* If yes is selected, these routines go to work.  Most of these are  *)
(* copied from the Set_Cluster_Point procedure with some changes.     *)

  if (Choice = #121) then
    begin
      SetWindow(2);
      Center_Text(8,'Place the cross-hair over the');
      Center_Text(9,'pixel with the desired       ');
      Center_Text(10,'threshold value and press    ');
      Center_Text(11,'ENTER.  The altered cluster  ');
      Center_Text(12,'area will appear in this     ');
      Center_Text(13,'window.                      ');
      SaveWorkSection(1,0,0,8*lw,5*lh);
      SetFillStyle(SolidFill,0);
      Bar(0,0,8*lw,5*lh);
      SetWindow(1);
      pixel := GetPixel(ex[active],ey[active]);
      Mark_Point(ex[active],ey[active],3,pixel);
      Show_Coordinates;
      repeat
      choice := Upcase(ReadKey);
      if choice = #00 then
      begin
      choice := ReadKey;
```

```
      if choice in [Up,Dn,Rt,Lt] then
       begin
       Unmark_Point(ex[active],ey[active],3);
       case choice of
         Up : ey[active] := IMax(0,ey[active]-step);
         Dn : ey[active] := IMin(ey[active]+step,cy-1);
         Lt : ex[active] := IMax(0,ex[active]-step);
         Rt : ex[active] := IMin(ex[active]+step,cx-1);
         end; {case}
       pixel := GetPixel(ex[active],ey[active]);
       Mark_Point(ex[active],ey[active],3,pixel);
       Show_Coordinates;
       end {if}
     else if choice in [PgUp,PgDn] then
       case choice of
         PgUp : step := IMin(64,step shl 1);
         PgDn : step := IMax(1,step shr 1);
         end; {case}
     end;{else}
  until choice = CR;
  SetWindow(2);
  ClearViewPort;
  SetWindow(1);
  SetWriteMode(XORPut);
  Rectangle(xs,ys,xs+arg,ys+arg);
  Unmark_Point(ex[active],ey[active],3);

(* After the threshold pixel is set, these routines will take the     *)
(* value of the threshold pixel and use it to determine which pixels  *)
(* within the cluster area will be set to a brightness of zero.  The  *)
(* altered cluster area will appear in Window Two so it can be used   *)
(* to compare with the results of the clustering routines in Window   *)
(* Three.                                                             *)

  for j := 0 to arg-1 do
    for i := 0 to arg-1 do
      begin
        pix := GetPixel(xs+i,ys+j);
        if (pix < pixel) then
          begin
            SetWindow(2);
            PutPixel(xs+i,ys+j,0);
            SetWindow(1);
          end
        else
          begin
            SetWindow(2);
            PutPixel(xs+i,ys+j,pix);
            SetWindow(1);
          end
      end; {for i}
  RestoreWorkSection(1,0,0,8*lw,5*lh);
  SetWindow(1);
  Goto interrupt1;
```

```
  end;

Beep(500);
Goto interrupt2;

interrupt1:
  menu_active := false;
  submenu_active := false;
  end, {Cluster_Histogram}


(* This is the main procedure of the clustering program.  From this  *)
(* procedure, all other procedures necessary to run the clustering   *)
(* algorithms are called.                                            *)

Procedure Cluster_Analysis (arg : byte);

LABEL

   ExitProg,interrupt2,interrupt3,interrupt4,interrupt5,interrupt6,
   interrupt7,interrupt8,interrupt9,interrupt10,interrupt11,
   interrupt12,interrupt13,ExitProg1,Finish;

VAR

   simptr,simptr2              : ARRDESP;
   Xptr,Yptr                   : ARRDESP;
   ptr2,Visptr,IRptr           : ARRDESP;
   nrows,ncols                 : Longint;
   Simval                      : Realtype;
   err                         : integer;
   pixel1,pixel2               : byte;
   arg2,m,n                    : byte;
   i,j,xs,ys,a,b               : integer;
   xpos,ypos,c,d,e,f           : integer;
   xpos2,ypos2                 : integer;
   VisWeight,IRWeight          : real;
   XWeight,YWeight             : real;
   VisPart,IRPart              : realtype;
   choice,choice1,choice2      : char;
   choice3,ch                  : char;
   Counter,chce3               : string[5];
   Result                      : real;
   DiffX, DiffY                : realtype;
   TempResult                  : real;
   minvalue,maxvalue           : integer;
   Flag,Flag2                  : integer;
   MatRows,MatCols             : longint;
   ComRows,ComCols             : longint;
   Iter, Numiter               : longint;
   Cur_Col_Pos                 : integer;
   Cur_Row_Pos                 : integer;
   Cur_Sim_Val                 : realtype;
   Next_Sim_Val                : realtype;
```

```
   Last_Sim_Val                  : realtype;
   ClusterNum                    : integer;
   Current_Val,Next_Val          : integer;
   Mean_X,Mean_Y                 : realtype;
   Total_Vis,Total_IR            : integer;
   Number_Pixels                 : realtype;
   Mean_Vis,Mean_IR              : realtype;
   Total_StandDev_Vis            : realtype;
   Total_StandDev_IR             : realtype;
   StandDev_X,StandDev_Y         : realtype;
   StandDev_Vis,StandDev_IR      : realtype;
   Z_Vis,Z_IR,Z_X,Z_Y            : realtype;
   Done                          : Boolean;
   Parray                        : array[0..149,0..149,0..1] of byte;

   Cluster_One_IR,Cluster_Two_IR                              : realtype;
   Cluster_One_Vis,Cluster_Two_Vis                            : realtype;
   Cluster_One_X,Cluster_One_Y,Cluster_Two_X,Cluster_Two_Y    : realtype;
   IR_Cluster_One_Val,IR_Cluster_Two_Val                      : realtype;
   Vis_Cluster_One_Val,Vis_Cluster_Two_Val                    : realtype;
   X_Cluster_One_Val,X_Cluster_Two_Val                        : realtype;
   Y_Cluster_One_Val,Y_Cluster_Two_Val                        : realtype;
   Cluster_One_Size,Cluster_Two_Size                          : longint;

begin
arg2 := arg div 2;
xy[active] := IMax(arg2,xy[active]);     (* Set the positioning       *)
xy[active] := IMin(xy[active],cy-arg2);  (* variables used to locate  *)
xx[active] := IMin(xx[active],cx-arg2);  (* keep track of the         *)
xx[active] := IMax(arg2,xx[active]);     (* clusteing area.           *)
MoveCursor_Top;
Select_Cluster_Box(arg,choice);  (* Call to procedure to allow user   *)
if choice <> ESC then            (* to choose where the clustering    *)
   begin                         (* area will be.                     *)
   a := 0;
   b := 0;
   xs := xx[active] - arg2;      (* Upper left, and lower right corner *)
   ys := xy[active] - arg2;      (* of the clustering area.           *)

   Mark_Cluster_Box(xx[active],xy[active],arg);
    for m := 0 to 149 do         (* Set all values in Parray to zero  *)
     for n := 0 to 149 do
      begin
         Parray[m,n,0] := 0;
         Parray[m,n,1] := 0;
      end; {for n}
   for j := ys to ys + arg - 1 do      (* Read visible pixel values  *)
       for i := xs to xs + arg - 1 do  (* into Parray.               *)
          begin
          pixel1 := GetPixel(i,j);
          PutPixel(i,j,255);
          a := i - xs;
          b := j - ys;
          Parray[a,b,0] := pixel1;
```

```
          PutPixel(i,j,pixel1);
        SetWindow(2);                    (* Read infrared pixel values *)
          pixel2 := GetPixel(i,j);       (* into Parray.               *)
          PutPixel(i,j,255);
          Parray[a,b,1] := pixel2;
          PutPixel(i,j,pixel2);
        SetWindow(1);
          end; {for i}
 end {if}
 else
   Goto ExitProg;

(* This section of code displays the main clustering menu.            *)

SetWindow(4);
interrupt6:
ClearViewPort;
   Center_Text(1, 'Select which option you wish ..... ');
   Center_Text(2, '   ');
   Center_Text(3, '0 : Cluster using Similarity Matrix ');
   Center_Text(4,'     and User-Defined Seed Points     ');
   Center_Text(5, '   ');
   Center_Text(6, '1 : Cluster Using Similarity Matrix ');
   Center_Text(7, '    and Automatic Seed Points       ');
   Center_Text(8, '   ');
   Center_Text(9, '2 : Show Ranges of Similarity Values');
   Center_Text(10, '    from One User-Defined Seed Point');
   Center_Text(11, '  ');
   Center_Text(12, '3 : Option 0 With Data Normalization ');
   Center_Text(13, '  ');
   Center_Text(14, '4 : Option 1 With Data Normalization ');
   Center_Text(15, '  ');
   Center_Text(16, '5 : Histogram of Clustering Area     ');
   Center_Text(17,'   ');
   Center_Text(18, 'ESC : Exit                          ');
   Choice := Readkey;
   if choice = #48 then
      begin
        Flag := 0;
        Flag2 := 0;
        Goto interrupt2;
      end
   else if choice = #49 then
      begin
        Flag := 1;
        Flag2 := 0;
        Goto interrupt2;
      end
   else if choice = #50 then
      begin
        Flag := 2;
        Flag2 := 2;
        Goto interrupt2;
      end
```

```
    else if choice = #51 then
       begin
         Flag := 0;
         Flag2 := 1;
         Goto interrupt2;
       end
    else if choice = #52 then
       begin
         Flag := 1;
         Flag2 := 1;
       end
    else if choice = #53 then
       begin
         SetWindow(1);
         Cluster_Histogram(xs,ys,arg);  (* Call to histogram procedure *)
         Repeat
           ch := ReadKey;
           Case Ch of
            #27 : Done := True;
           end {Case}
         Until Done = True;
         Goto Finish;
       end
    else if choice = ESC then  (* If ESC selected, goto end of program *)
       begin
         Goto Finish;
       end
    else
       begin
         Beep(500);
         Goto interrupt6;
       end;

(* The next section of code allows the user to select the weights     *)
(* for each of the categories in the distance equation.               *)

interrupt2:
   VisWeight := 0.0;
   IRWeight := 0.0;
   XWeight := 0.0;
   YWeight := 0.0;
   SetWindow(4);
   ClearViewPort;
     Center_Text(1, 'Select the weights you wish to give to');
     Center_Text(2, 'the variables used to calculate the   ');
     Center_Text(3, 'similarity values                     ');
     Center_Text(4, '                                      ');
     Center_Text(5, '0 : Vis/IRPart = 1      Diff X/Y = 1  ');
     Center_Text(6, '1 : Vis/IRPart = 1      Diff X/Y = .5 ');
     Center_Text(7, '2 : Vis/IRPart = 1      Diff X/Y = .25');
     Center_Text(8, '3 : Vis/IRPart = 1      Diff X/Y = .10');
     Center_Text(9, '4 : Vis/IRPart = 1      Diff X/Y = 0  ');
     Center_Text(10, '                                     ');
     Center_Text(11, '5 : Vis/IRPart = .5     Diff X/Y = 1  ');
```

```
    Center_Text(12, '6 : Vis/IRPart = .25    Diff X/Y = 1  ');
    Center_Text(13, '7 : Vis/IRPart = .1     Diff X/Y = 1  ');
    Center_Text(14, '8 : Vis/IRPart = 0      Diff X/Y = 1  ');
    Center_Text(15, '                                      ');
    Center_Text(16, '9 : Vis/IRPart = 1      X = .1  Y = 1 ');
    Center_Text(17, 'a : Vis/IRPart = 1      X = 1   Y = .1');
    Center_Text(18, '                                      ');
    Center_Text(19, 'u : Input specific values             ');
    choicel := ReadKey;
if choicel = #117 then
    begin
      ClearViewPort;
      Center_Text(1, 'Enter the values for each variable   ');
      Center_Text(2, 'in the following order.  Press ENTER ');
      Center_Text(3, 'after entering each number.          ');
      Center_Text(4, '                                     ');
      Center_Text(5, 'Visible Weight        (0 - 1.0)      ');
      Center_Text(6, 'Infrared Weight       (0 - 1.0)      ');
      Center_Text(7, 'X Position Weight     (0 - 1.0)      ');
      Center_Text(8, 'Y Position Weight     (0 - 1.0)      ');
      OutTextXY(40,86,'= Visible Weight');
      OutTextXY(40,101,'= IR Weight');
      OutTextXY(40,116,'= X Weight');
      OuttextXY(40,131,'= Y Weight');
    MoveCursor;
    ReadLn(VisWeight,IRWeight,XWeight,YWeight);
    MoveCursor_Top;
    end
else if choicel = #48 then
    begin
      VisWeight := 1;
      IRWeight  := 1;
      XWeight   := 1;
      YWeight   := 1;
    end
else if choicel = #49 then
    begin
      VisWeight := 1;
      IRWeight  := 1;
      XWeight   := 0.5;
      YWeight   := 0.5;
    end
else if choicel = #50 then
    begin
      VisWeight := 1;
      IRWeight  := 1;
      XWeight   := 0.25;
      YWeight   := 0.25;
    end
else if choicel = #51 then
    begin
      VisWeight := 1;
      IRWeight  := 1;
      XWeight   := 0.1;
```

```
      YWeight   := 0.1;
    end
else if choicel = #52 then
    begin
      VisWeight := 1;
      IRWeight  := 1;
      XWeight   := 0;
      YWeight   := 0;
    end
else if choicel = #53 then
    begin
      VisWeight := 0.5;
      IRWeight  := 0.5;
      XWeight   := 1;
      YWeight   := 1;
    end
else if choicel = #54 then
    begin
      VisWeight := 0.25;
      IRWeight  := 0.25;
      XWeight   := 1;
      YWeight   := 1;
    end
else if choicel = #55 then
    begin
      VisWeight := 0.1;
      IRWeight  := 0.1;
      XWeight   := 1;
      YWeight   := 1;
    end
else if choicel = #56 then
    begin
      VisWeight := 0;
      IRWeight  := 0;
      XWeight   := 1;
      YWeight   := 1;
    end
else if choicel = #57 then
    begin
      VisWeight := 1;
      IRWeight  := 1;
      XWeight   := 1;
      YWeight   := 0.1;
    end
else if choicel = #97 then
    begin
      VisWeight := 1;
      IRWeight  := 1;
      XWeight   := 0.1;
      YWeight  := 1;
    end
else
    begin
      Beep(500);
```

```
      Goto interrupt2;
    end;

(* This section of code determines which of the clustering options   *)
(* was selected by the setting of the Flag variables.  Depending on  *)
(* the option, the appropriate calls to the seed point selection     *)
(* procedures are executed.                                          *)

if (Flag = 0) and (Flag2 = 0) then
  begin
    SetWindow(4);
    ClearViewPort;
    Center_Text(7,'Select the first seed point to be ');
    Center_Text(8, a pixel within a cloud region.    ');
    SetWindow(1);
    Set_Cluster_Point(xs,ys,arg,xpos,ypos);
    Mark_Cluster_Box(xx[active],xy[active],arg);
    SetWindow(4);
    ClearViewport;
    Center_Text(7,'Select the second seed point to   ');
    Center_Text(8,'be a pixel in a non-cloud region. ');
    SetWindow(1);
    Set_Cluster_Point(xs,ys,arg,xpos2,ypos2);
    Goto interrupt7;
  end; {if}
if (Flag = 1) and (Flag2 = 0) then
  begin
    Current_Val := -1;
    for b := 0 to arg-1 do          (* Automatic selection of brightest *)
      for a := 0 to arg-1 do        (* pixel in clustering area.        *)
        begin
          Next_Val := Parray[a,b,0];
          if (Next_Val > Current_Val) then
            begin
              Current_Val := Next_Val;
              xpos := a+xs;
              ypos := b+ys;
            end
        end; {for a}
    Current_Val := 300;
    for b:= 0 to arg-1 do           (* Automatic selection of dimmest   *)
      for a := 0 to arg-1 do        (* pixel in clustering area.        *)
        begin
          Next_Val := Parray[a,b,0];
          if (Next_Val < Current_Val) then
            begin
              Current_Val := Next_Val;
              xpos2 := a+xs;
              ypos2 := b+ys;
            end
        end; {for a}
    Goto interrupt7;
  end; {if}
if (Flag = 0) and (Flag2 = 1) then
```

```
  begin
    SetWindow(4);
    ClearViewPort;
    Center_Text(7,'Select the first seed point to be a ');
    Center_Text(8,'pixel within a cloud region.        ');
    SetWindow(1);
    Set_Cluster_Point(xs,ys,arg,xpos,ypos);
    Mark_cluster_Box(xx[active],xy[active],arg);
    SetWindow(4);
    ClearViewport;
    Center_Text(7,'Select the second seed point to be  ');
    Center_Text(8,'a pixel in a non-cloud region.      ');
    SetWindow(1);
    Set_Cluster_Point(xs,ys,arg,xpos2,ypos2);
    Goto interrupt12;
  end; {if}
if (Flag = 1) and (Flag2 = 1) then
  begin
    Current_Val := -1;
    for b := 0 to arg-1 do          (* Automatic selection of brightest *)
      for a := 0 to arg-1 do        (* pixel in clustering area.        *)
        begin
          Next_Val := Parray[a,b,0];
          if (Next_Val > Current_Val) then
            begin
              Current_Val := Next_Val;
              xpos := a+xs;
              ypos := b+ys;
            end
        end; {for a}
    Current_Val := 300;
    for b:= 0 to arg-1 do           (* Automatic selection of dimmest   *)
      for a := 0 to arg-1 do        (* pixel in clustering area.        *)
        begin
          Next_Val := Parray[a,b,0];
          if (Next_Val < Current_Val) then
            begin
              Current_Val := Next_Val;
              xpos2 := a+xs;
              ypos2 := b+ys;
            end
        end; {for a}
    Goto interrupt12;
  end; {if}
if (Flag = 2) and (Flag2 = 2) then
  begin
    SetWindow(4);
    ClearViewPort;
    Center_Text(7,'Select the seed point');
    SetWindow(1);
    Set_Cluster_Point(xs,ys,arg,xpos,ypos);
  end;
```

```
(* This section of code is used to determine what range of values to  *)
(* display when Option Two is selected from the main clustering menu. *)

SetWindow(3);
ClearViewPort;
interrupt4:
SetWindow(4);
   DiffX := 0.0;
   DiffY := 0.0;
   VisPart := 0.0;
   IRPart := 0.0;
   TempResult := 0.0;
   Result := 0.0;
ClearViewPort;
     Center_Text(1,' Select the range for the similarity ');
     Center_Text(2,' value (S) you wish to display       ');
     Center_Text(3,'                                     ');
     Center_Text(4,'0 :  0 <= S < 10  a : 100 <= S < 110');
     Center_Text(5,'1 : 10 <= S < 20  b : 110 <= S < 120');
     Center_Text(6,'2 : 20 <= S < 30  c : 120 <= S < 130');
     Center_Text(7,'3 : 30 <= S < 40  d : 130 <= S < 140');
     Center_Text(8,'4 : 40 <= S < 50  e : 140 <= S < 150');
     Center_Text(9,'5 : 50 <= S < 60  f : 150 <= S < 160');
     Center_Text(10,'6 : 60 <= S < 70  g : 160 <= S < 170');
     Center_Text(11,'7 : 70 <= S < 80  h : 170 <= S < 180');
     Center_Text(12,'8 : 80 <= S < 90  i : 180 <= S < 190');
     Center_Text(13,'9 : 90 <= S < 100 j : 190 <= S < 200');
     Center_Text(14,'                  k : 200 <= S < 210');
     Center_Text(15,'                                    ');
     Center_Text(16,'   p : Change Palette               ');
     Center_Text(17,'   w : Change the Weights           ');
     Center_Text(18,' TAB : Blank Window 3               ');
     Center_Text(19,' ');
     Center_Text(20,' ESC : Quit                         ');
     choice2 := ReadKey;
SetWindow(3);
  if choice2 = #9 then              (* Select TAB to clear window 3 *)
     begin
       ClearViewPort;
       GoTo interrupt4;
     end
  else if choice2 = ESC then        (* Select ESC to exit the routine *)
     begin
       Goto ExitProg;
     end
  else if choice2 = #112 then       (* Select p to change the palette *)
     begin
       Palettes;
       GoTo interrupt4;
     end
  else if choice2 = #119 then       (* Select w to change the weights *)
     begin
       GoTo interrupt2;
     end
```

```
else if choice2 = #48 then          (* Set weights based upon menu    *)
 begin                              (* selection.                     *)
    minvalue := 0;
    maxvalue := 10;
    Goto interrupt5;
 end
else if choice2 = #49 then
 begin
    minvalue := 10;
    maxvalue := 20;
    Goto interrupt5;
 end
else if choice2 = #50 then
 begin
    minvalue := 20;
    maxvalue := 30;
    Goto interrupt5;
 end
else if choice2 = #51 then
 begin
    minvalue := 30;
    maxvalue := 40;
    Goto interrupt5;
 end
else if choice2 = #52 then
 begin
    minvalue := 40;
    maxvalue := 50;
    Goto interrupt5;
 end
else if choice2 = #53 then
 begin
    minvalue := 50;
    maxvalue := 60;
    Goto interrupt5;
 end
else if choice2 = #54 then
 begin
    minvalue := 60;
    maxvalue := 70;
    Goto interrupt5;
 end
else if choice2 = #55 then
 begin
    minvalue := 70;
    maxvalue := 80;
    Goto interrupt5;
 end
else if choice2 = #56 then
 begin
    minvalue := 80;
    maxvalue := 90;
    Goto interrupt5;
 end
```

```
else if choice2 = #57 then
 begin
    minvalue := 90;
    maxvalue := 100;
    Goto interrupt5;
 end
else if choice2 = #97 then
 begin
    minvalue := 100;
    maxvalue := 110;
    Goto interrupt5;
 end
else if choice2 = #98 then
 begin
    minvalue := 110;
    maxvalue := 120;
    Goto interrupt5;
 end
else if choice2 = #99 then
 begin
    minvalue := 120;
    maxvalue := 130;
    Goto interrupt5;
 end
else if choice2 = #100 then
 begin
    minvalue := 130;
    maxvalue := 140;
    Goto interrupt5;
 end
else if choice2 = #101 then
 begin
    minvalue := 140;
    maxvalue := 150;
    Goto interrupt5;
 end
else if choice2 = #102 then
 begin
    minvalue := 150;
    maxvalue := 160;
    Goto interrupt5;
 end
else if choice2 = #103 then
 begin
    minvalue := 160;
    maxvalue := 170;
    Goto interrupt5;
 end
else if choice2 = #104 then
 begin
    minvalue := 170;
    maxvalue := 180;
    Goto interrupt5;
 end
```

```
  else if choice2 = #105 then
   begin
      minvalue := 180;
      maxvalue := 190;
      Goto interrupt5;
   end
  else if choice2 = #106 then
   begin
      minvalue := 190;
      maxvalue := 200;
      Goto interrupt5;
   end
  else if choice2 = #107 then
   begin
      minvalue := 200;
      maxvalue := 210;
      Goto interrupt5;
   end
  else
   begin
     Beep(500);
     Goto interrupt4;
   end;

(* This section of code does the actual calculations to determine    *)
(* which pixels have distance values within the range selected to be *)
(* displayed.                                                        *)

interrupt5:
   PutPixel(xpos,ypos,255);
   for j := ys to ys+arg-1 do
      for i := xs to xs+arg-1 do
      begin
      a := i - xs;
      b := j - ys;
      c := xpos - xs;  (* c : x position of seed pixel *)
      d := ypos - ys;  (* d : y position of seed pixel *)

(* This code is the implementation of the distance formula.          *)

      DiffX :=  XWeight * SQR(c - a);
      DiffY :=  YWeight * SQR(d - b);
      VisPart := VisWeight * SQR(Parray[c,d,0] - Parray[a,b,0]);
      IRPart  := IRWeight * SQR(Parray[c,d,1] - Parray[a,b,1]);
         TempResult := (VisPart + IRPart + DiffX + DiffY);
         Result  := SQRT(TempResult);
         if (minvalue <= Result) and (Result < maxvalue) then
           begin
             PutPixel(a+xs,b+ys,Parray[a,b,0]);
           end; {if}
      end; {for a}
GoTo interrupt4;       (* Loop back to select next range to display . *)
```

```
(* This is the start of the routines used to cluster the pixels from  *)
(* two seed points using the centroid method without data             *)
(* normalization.                                                     *)

interrupt7:
  SetWindow(3);
  ClearViewPort;
  PutPixel(xpos,ypos,255);    (* Turn on the two seed pixels in Window *)
  PutPixel(xpos2,ypos2,150); (* Three.                                 *)

  if (arg = 20) then        (* These statements set the number of rows *)
   begin                    (* and columns used by the similarity and *)
    nrows := 20;            (* tracking matrix.                       *)
    ncols := 20;
   end
  else if (arg = 50) then
   begin
    nrows := 50;
    ncols := 50;
   end
  else if (arg = 100) then
   begin
    nrows := 100;
    ncols := 100;
   end
  else
   begin
    nrows := 150;
    ncols := 150;
   end;

  simptr := HMatDef(nrows,ncols);     (* Create the similarity matrix  *)
  if (simptr = nil) then              (* for the first seed point.     *)
    begin
      Writeln ('Cannot Create Matrix');
      EXIT;
    end;

  simptr2 := HMatDef(nrows,ncols);    (* Create the similarity matrix  *)
  if (simptr = nil) then              (* for the second seed point.    *)
    begin
      Writeln ('Cannot Create Matrix');
      EXIT;
    end;

  ptr2 := HMatDef(nrows,ncols);          (* Create the tracking matrix *)
  if (ptr2 = nil) then
    begin
      Writeln ('Cannot Create Comparison Matrix');
      EXIT;
    end;

(* Fill Matrix with Similarity Values *)
```

```
  for b := 0 to arg-1 do
    for a := 0 to arg-1 do
      begin
      c := xpos - xs;   (* c : x position of first seed pixel *)
      d := ypos - ys;   (* d : y position of first seed pixel *)
      e := xpos2 - xs;  (* e : x position of second seed pixel *)
      f := ypos2 - ys;  (* f : y position of second seed pixel *)

(* Similarity value calculations using Euclidean distance formula.    *)
(* This section is the distance calculations to the first seed pixel. *)

      DiffX :=  XWeight * SQR(c - a);
      DiffY :=  YWeight * SQR(d - b);
      VisPart := VisWeight * SQR(Parray[c,d,0] - Parray[a,b,0]);
      IRPart  := IRWeight * SQR(Parray[c,d,1] - Parray[a,b,1]);
        TempResult := (VisPart + IRPart + DiffX + DiffY);
        SimVal := SQRT(TempResult);
        HMatWrtEl (simptr,a,b,SimVal,err);

(* This section is the distance calculations to the second seed pixel.*)

      DiffX := XWeight * SQR(e - a);
      DiffY := YWeight * SQR(f - b);
      VisPart := VisWeight * SQR(Parray[e,f,0] - Parray[a,b,0]);
      IRPart  := IRWeight * SQR(Parray[e,f,1] - Parray[a,b,1]);
        TempResult := (VisPart + IRPart + DiffX + DiffY);
        SimVal := SQRT(TempResult);
        HMatWrtEl (simptr2,a,b,SimVal,err);
      end; {for a}

(* Initialize the cluster values with the seed pixel's values *)

Cluster_One_IR := Parray[xpos-xs,ypos-ys,1];
Cluster_One_Vis := Parray[xpos-xs,ypos-ys,0];
Cluster_One_X := xpos-xs;
Cluster_One_Y := ypos-ys;
IR_Cluster_One_Val := Parray[xpos-xs,ypos-ys,1];
Vis_Cluster_One_Val := Parray[xpos-xs,ypos-ys,0];
X_Cluster_One_Val := xpos-xs;
Y_Cluster_One_Val := ypos-ys;

Cluster_Two_IR := Parray[xpos2-xs,ypos2-ys,1];
Cluster_Two_Vis := Parray[xpos2-xs,ypos2-ys,0];
Cluster_Two_X := xpos2-xs;
Cluster_Two_Y := ypos2-ys;
IR_Cluster_Two_Val := Cluster_Two_IR;
Vis_Cluster_Two_Val := Cluster_Two_Vis;
X_Cluster_Two_Val := Cluster_Two_X;
Y_Cluster_Two_Val := Cluster_Two_Y;


(* This section of code determines how many iterations the looping    *)
(* process will complete.                                             *)
```

```
SetWindow(4);
interrupt8:
ClearViewPort;
Center_Text(1, 'Select the number of iterations');
Center_Text(2, '                                 ');
Center_Text(3, '                                 ');
Center_Text(4, '0 : 5              3 : 100       ');
Center_Text(5, '1 : 10             4 : 398       ');
Center_Text(6, '2 : 50             5 : 2498      ');
Center_Text(7, '                                 ');
Center_Text(8, 'u : Enter # of iterations        ');
Center_Text(9, '                                 ');
Center_Text(10, 'ESC : exit                       ');
Choice3 := ReadKey;
if choice3 = #48 then
   Numiter := 5
else if choice3 = #49 then
   Numiter := 10
else if choice3 = #50 then
   Numiter := 50
else if choice3 = #51 then
   Numiter := 100
else if choice3 = #52 then
   Numiter := 398
else if choice3 = #53 then
   Numiter := 2498
else if choice3 = ESC then
   goto interrupt3
else if choice3 = #117 then
   begin
     ClearViewPort;
     Center_Text(1, 'Enter the number of iterations');
     Center_Text(2, 'Press ENTER when complete     ');
     MoveCursor;
     ReadLn(Numiter);
     MoveCursor_Top;
   end
else
  begin
   Beep(500);
   goto interrupt8;
  end;
Str(Numiter,choice3);
OutTextXY(35,110,'Number of iterations =');
OutTextXY(218,110,choice3);
SetWindow(3);
Cluster_One_Size := 1;                    (* Each cluster initially   *)
Cluster_Two_Size := 1;                    (* has one pixel.           *)
HMatWrtEl (ptr2,xpos-xs,ypos-ys,1.0,err);
HMatWrtEl (ptr2,xpos2-xs,ypos2-ys,2.0,err);

(* This is where the looping process begins.                          *)

For Iter := 1 to Numiter do
```

```
(* First thing to do is search for the smallest distance value.       *)

 begin
  Cur_Sim_Val := 1000.0;
  for b := 0 to arg-1 do          (* Search the first distance array. *)
    for a := 0 to arg-1 do
      begin
        if (HMatReadEl(ptr2,a,b,err) < 1.0) then
          begin
            Next_Sim_Val := HMatReadEl (simptr,a,b,err);
            if (Next_Sim_Val < Cur_Sim_Val) then
              begin
                Cur_Sim_Val := Next_Sim_Val;
                Cur_Col_Pos := a;
                Cur_Row_Pos := b;
                ClusterNum := 1;
              end
        end
      end; {for a}
  for b := 0 to arg-1 do          (* Search the second distance array.*)
    for a := 0 to arg-1 do
      begin
        if (HMatReadEl(ptr2,a,b,err) < 1.0) then
          begin
            Next_Sim_Val := HMatReadEl (simptr2,a,b,err);
            if (Next_Sim_Val < Cur_Sim_Val) then
              begin
                Cur_Sim_Val := Next_Sim_Val;
                Cur_Col_Pos := a;
                Cur_Row_Pos := b;
                ClusterNum := 2;
              end
          end
    end; {for a}


(* The cluster corresponding to the array with the smallest value is  *)
(* incremented by one.  The pixel is turned on in Window Three in the *)
(* appropriate brightness.  If it is a cloud, it will use the same    *)
(* value as in the visible image.  If it isn't a cloud, it will be    *)
(* set to a brightness of 50.                                         *)

  if ClusterNum = 1 then
  begin
    HMatWrtEl (ptr2,Cur_Col_Pos, Cur_Row_Pos,1.0,err);
    Cluster_One_Size := Cluster_One_Size + 1;
    PutPixel(Cur_Col_Pos+xs,Cur_Row_Pos+ys,
             Parray[Cur_Col_Pos,Cur_Row_Pos,0]);
  end;

if ClusterNum = 2 then
  begin
    HMatWrtEl (ptr2,Cur_Col_Pos,Cur_Row_Pos,2.0,err);
    Cluster_Two_Size := Cluster_Two_Size + 1;
```

```
    PutPixel(Cur_Col_Pos+xs,Cur_Row_Pos+ys,50);
  end;

(* Display the iteration number currently underway in Window Four.    *)

  Str(Iter, Counter);
  SetWindow(4);
  OutTextXY(35,130,'Iterations Completed =');
  SetColor(0);
  OutTextXY(218,130,#219);
  OutTextXY(225,130,#219);
  OutTextXY(232,130,#219);
  OutTextXY(239,130,#219);
  OutTextXY(246,130,#219);
  OutTextXY(253,130,#219);
  SetColor(255);
  OutTextXY(218,130,Counter);
  SetWindow(3);

(* Compute the new values for the clusters.                           *)

if ClusterNum = 1 then
 begin
  Cluster_One_IR := Cluster_One_IR + Parray[Cur_Col_Pos,Cur_Row_Pos,1];
  Cluster_One_Vis :=Cluster_One_Vis+ Parray[Cur_Col_Pos,Cur_Row_Pos,0];
  Cluster_One_X := Cluster_One_X + Cur_Col_Pos;
  Cluster_One_Y := Cluster_One_Y + Cur_Row_Pos;

  IR_Cluster_One_Val := Cluster_One_IR/Cluster_One_Size;
  Vis_Cluster_One_Val := Cluster_One_Vis/Cluster_One_Size;
  X_Cluster_One_Va' := Cluster_One_X/Cluster_One_Size;
  Y_Cluster_One_Val := Cluster_One_Y/Cluster_One_Size;
 end;

if ClusterNum = 2 then
 begin
  Cluster_Two_IR := Cluster_Two_IR + Parray[Cur_Col_Pos,Cur_Row_Pos,1];
  Cluster_Two_Vis :=Cluster_Two_Vis + Parray[Cur_Col_Pos,Cur_Row_Pos,0];
  Cluster_Two_X := Cluster_Two_X + Cur_Col_Pos;
  Cluster_Two_Y := Cluster_Two_Y + Cur_Row_Pos;

  IR_Cluster_Two_Val := Cluster_Two_IR/Cluster_Two_Size;
  Vis_Cluster_Two_Val := Cluster_Two_Vis/Cluster_Two_Size;
  X_Cluster_Two_Val := Cluster_Two_X/Cluster_Two_Size;
  Y_Cluster_Two_Val := Cluster_Two_Y/Cluster_Two_Size;
 end;

(* Begin the loop to fill the similarity matrices with updated values *)

if ClusterNum = 1 then
 begin
  for b := 0 to arg-1 do
    for a := 0 to arg-1 do
      begin
```

```
        if (HMatReadEl(ptr2,a,b,err) < 0.0) then
          begin
            DiffX := XWeight * SQR(X_Cluster_One_Val - a);
            DiffY := YWeight * SQR(Y_Cluster_One_Val - b);
            VisPart := VisWeight * SQR(Vis_Cluster_One_Val -
                       Parray[a,b,0]);
            IRPart := IRWeight * SQR(IR_Cluster_One_Val -
                      Parray[a,b,1]);
              TempResult := (VisPart + IRPart + DiffX + DiffY);
              SimVal := SQRT(TempResult);
              HMatWrtEl (simptr,a,b,SimVal,err);
          end
      end; {for r}
 end; {if}

(* Second similarity matrix *)

if ClusterNum = 2 then
 begin
  for b := 0 to arg-1 do
    for a := 0 to arg-1 do
      begin
        if (HMatReadEl(ptr2,a,b,err) < 0.0) then
          begin
            DiffX := XWeight * SQR(X_Cluster_Two_Val - a);
            DiffY := YWeight * SQR(Y_Cluster_Two_Val - b);
            VisPart := VisWeight * SQR(Vis_Cluster_Two_Val -
                       Parray[a,b,0]);
            IRPart := IRWeight * SQR(IR_Cluster_Two_Val -
                      Parray[a,b,1]);
              TempResult := (VisPart + IRPart + DiffX + DiffY);
              SimVal := SQRT(TempResult);
              HMatWrtEl (simptr2,a,b,SimVal,err);
          end
      end; {for a}
 end; {if}

(* This routine lets the user get out of the looping process by       *)
(* pressing the ESC key on the keyboard.                              *)

  if (keypressed = TRUE) then
    begin
      Choice := Readkey;
      if (Choice = ESC) then
        Goto interrupt3;
    end

end; {iter loop}     (* This is the end of the looping routines *)
Goto interrupt3;     (* Proceed to housekeeping routines        *)


(* This is the start of the routines which are used to do the         *)
(* the clustering from two seed points using the centroid method with *)
(* data normalization.                                                *)
```

```
interrupt12:
  SetWindow(3);
  ClearViewPort;
  PutPixel(xpos,ypos,255);    (* Turn on the seed pixels in Window     *)
  PutPixel(xpos2,ypos2,150); (* Three.                                 *)

  if (arg = 20) then        (* These statements set the number of rows *)
   begin                    (* and columns used by the similarity and *)
    nrows := 20;            (* tracking matrix.                        *)
    ncols := 20;
   end
  else if (arg = 50) then
   begin
    nrows := 50;
    ncols := 50;
   end
  else if (arg = 100) then
   begin
    nrows := 100;
    ncols := 100;
   end
  else
   begin
    nrows := 150;
    ncols := 150;
   end;

  simptr := HMatDef(nrows,ncols);      (* Create the similarity matrix *)
  if (simptr = nil) then
    begin
      Writeln ('Cannot Create Matrix');
      EXIT;
    end;

  simptr2 := HMatDef(nrows,ncols);    (* Create the second similarity  *)
  if (simptr = nil) then              (* matrix.                       *)
    begin
      Writeln ('Cannot Create Matrix');
      EXIT;
    end;

  ptr2 := HMatDef(nrows,ncols);          (* Create the tracking matrix *)
  if (ptr2 = nil) then
    begin
      Writeln ('Cannot Create Comparison Matrix');
      EXIT;
    end;

  Visptr := HMatDef(nrows,ncols);    (* Create the matrix to store the *)
  if (Visptr=nil) then               (* z-values for the visible data  *)
    begin
      Writeln ('Cannot create Visptr');
      Exit;
    end;
```

```
  IRptr := HMatDef(nrows,ncols);     (* Create the matrix to store the *)
  if (IRptr=nil) then                (* z-values for the infrared data *)
    begin
      Writeln ('Cannot create IRptr');
      Exit;
    end;

  Xptr := HMatDef(nrows,ncols);      (* Create the matrix to store the *)
  if (Xptr=nil) then                 (* z-values for the x positions   *)
    begin
      Writeln ('Cannot create Xptr');
      Exit;
    end;

  Yptr := HMatDef(nrows,ncols);      (* Create the matrix to store the *)
  if (Yptr=nil) then                 (* z-values for the y positions   *)
    begin
      Writeln ('Cannot create Yptr');
      Exit;
    end;

(* Normalize the raw category data using the standard deviation.      *)
(* First calculate the means of the four categories.                  *)

Mean_X := (arg-1)/2;
Mean_Y := Mean_Y;
Total_Vis := 0;
Total_IR := 0;
Number_Pixels := arg*arg;

for b := 0 to arg-1 do
  for a := 0 to arg-1 do
    begin
      Total_Vis := Total_Vis+Parray[a,b,0];
      Total_IR  := Total_IR +Parray[a,b,1];
    end; {for b}

Mean_Vis := Total_Vis/Number_Pixels;
Mean_IR  := Total_IR/Number_Pixels;

(* Now calculate the standard deviations. *)

Total_StandDev_Vis := 0.0;
Total_StandDev_IR  := 0.0;
if arg = 20 then
  begin
    StandDev_X := 5.916079783;
    StandDev_Y := StandDev_X;
  end
else if arg = 50 then
  begin
    StandDev_X := 14.57737974;
    StandDev_Y := StandDev_X;
  end
```

```
else if arg = 100 then
  begin
    StandDev_X := 29.01149198;
    StandDev_Y := StandDev_X;
  end
else if arg = 150 then
  begin
    StandDev_X := 43.44536799;
    StandDev_Y := StandDev_X;
  end
else
  begin
    Goto interrupt3;
  end;

for b := 0 to arg-1 do
 for a := 0 to arg-1 do
  begin
  Total_StandDev_Vis := Total_StandDev_Vis+SQR(Parray[a,b,0]-Mean_Vis);
  Total_StandDev_IR  := Total_StandDev_IR +SQR(Parray[a,b,1]-Mean_IR);
  end; {for b}

StandDev_Vis := SQRT(Total_StandDev_Vis/(Number_Pixels-1));
StandDev_IR  := SQRT(Total_StandDev_IR/(Number_Pixels-1));

(* Calculate the z-values and fill in the matrices. *)

for b := 0 to arg-1 do
 for a := 0 to arg-1 do
  begin
   Z_Vis := (Parray[a,b,0]-Mean_Vis)/StandDev_Vis;
   Z_IR  := (Parray[a,b,1]-Mean_IR)/StandDev_IR;
   Z_X   := (a-Mean_X)/StandDev_X;
   Z_Y   := (b-Mean_Y)/StandDev_Y;
   HMatWrtEl(Visptr,a,b,Z_Vis,err);
   HMatWrtEl(IRptr,a,b,Z_IR,err);
   HMatWrtEl(Xptr,a,b,Z_X,err);
   HMatWrtEl(Yptr,a,b,Z_Y,err);
  end; {for a}

(* Initialize the two clusters with the seed pixel's values. *)

Cluster_One_IR := HMatReadEl(IRptr,xpos-xs,ypos-ys,err);
Cluster_One_Vis := HMatReadEl(Visptr,xpos-xs,ypos-ys,err);
Cluster_One_X := HMatReadEl(Xptr,xpos-xs,ypos-ys,err);
Cluster_One_Y := HMatReadEl(Yptr,xpos-xs,ypos-ys,err);
IR_Cluster_One_Val := Cluster_One_IR;
Vis_Cluster_One_Val := Cluster_One_Vis;
X_Cluster_One_Val := Cluster_One_X;
Y_Cluster_One_Val := Cluster_One_Y;

Cluster_Two_IR := HMatReadEl(IRptr,xpos2-xs,ypos2-ys,err);
Cluster_Two_Vis := HMatReadEl(Visptr,xpos2-xs,ypos2-ys,err);
Cluster_Two_X := HMatReadEl(Xptr,xpos2-xs,ypos2-ys,err);
```

```
Cluster_Two_Y := HMatReadEl(Yptr,xpos2-xs,ypos2-ys,err);
IR_Cluster_Two_Val := Cluster_Two_IR;
Vis_Cluster_Two_Val := Cluster_Two_Vis;
X_Cluster_Two_Val := Cluster_Two_X;
Y_Cluster_Two_Val := Cluster_Two_Y;

(* Fill Matrices with Similarity Values. *)

for b := 0 to arg-1 do
 for a := 0 to arg-1 do
  begin
   c :=xpos - xs;
   d :=ypos - ys;
   e :=xpos2 - xs;
   f :=ypos2 - ys;
   DiffX := XWeight * SQR(HMatReadEl(Xptr,c,d,err) -
            HMatReadEl(Xptr,a,b,err));
   DiffY := YWeight * SQR(HMatReadEl(Yptr,c,d,err) -
            HMatReadEl(Yptr,a,b,err));
   VisPart := VisWeight * SQR(HMatReadEl(Visptr,c,d,err) -
              HMatReadEl(Visptr,a,b,err));
   IRPart := IRWeight * SQR(HMatReadEl(IRptr,c,d,err) -
             HMatReadEl(IRptr,a,b,err));
     TempResult := (VisPart+IRPart+DiffX+DiffY);
     SimVal := SQRT(TempResult);
     HMatWrtEl(simptr,a,b,SimVal,err);

   DiffX := XWeight * SQR(HMatReadEl(Xptr,e,f,err) -
            HMatReadEl(Xptr,a,b,err));
   DiffY := YWeight * SQR(HMatReadEl(Yptr,e,f,err) -
            HMatReadEl(Yptr,a,b,err));
   VisPart := VisWeight * SQR(HMatReadEl(Visptr,e,f,err) -
              HMatReadEl(Visptr,a,b,err));
   IRPart := IRWeight * SQR(HMatReadEl(IRptr,e,f,err) -
             HMatReadEl(IRPtr,a,b,err));
     TempResult := (VisPart+IRPart+DiffX+DiffY);
     SimVal := SQRT (TempResult);
     HMatWrtEl(simptr2,a,b,SimVal,err);
  end; {for a}


(* This section of code allows the user to select the number of      *)
(* iterations the loop will go through before stopping.              *)

SetWindow(4);
interrupt13:
ClearViewPort;
Center_Text(1, 'Select the number of iterations');
Center_Text(2, '                                 ');
Center_Text(3, '                                 ');
Center_Text(4, '0 : 5              3 : 100   ');
Center_Text(5, '1 : 10             4 : 398   ');
Center_Text(6, '2 : 50             5 : 2498  ');
Center_Text(7, '                                 ');
```

```
Center_Text(8, 'u : Enter # of iterations      ');
Center_Text(9, '                               ');
Center_Text(10, 'ESC : exit                     ');
Choice3 := ReadKey;
if choice3 = #48 then
   Numiter := 5
else if choice3 = #49 then
   Numiter := 10
else if choice3 = #50 then
   Numiter := 50
else if choice3 = #51 then
   Numiter := 100
else if choice3 = #52 then
   Numiter := 398
else if choice3 = #53 then
   Numiter := 2498
else if choice3 = ESC then
   goto interrupt3
else if choice3 = #117 then
   begin
     ClearViewPort;
     Center_Text(1, 'Enter the number of iterations');
     Center_Text(2, 'Press ENTER when complete     ');
     MoveCursor;
     ReadLn(Numiter);
     MoveCursor_Top;
   end
else
  begin
   Beep(500);
   goto interrupt13;
  end;
Str(Numiter,chce3);
OutTextXY(35,110,'Number of iteraticns =');
OutTextXY(218,110,chce3);

SetWindow(3);
Cluster_One_Size := 1;                (* Each cluster starts out with *)
Cluster_Two_Size := 1;                (* one pixel in it.             *)
HMatWrtEl (ptr2,xpos-xs,ypos-ys,1.0,err);
HMatWrtEl (ptr2,xpos2-xs,ypos2-ys,2.0,err);

(* This is where the looping process begins.                          *)

for Iter := 1 to Numiter do

(* The first thing to do is search for the smallest distance value    *)
(* in both distance arrays.                                           *)

 begin
  Cur_Sim_Val := 1000.0;
  for b := 0 to arg-1 do           (* Search the first array *)
    for a := 0 to arg-1 do
      begin
```

```
        if (HMatReadEl(ptr2,a,b,err) < 1.0) then
          begin
            Next_Sim_Val := HMatReadEl (simptr,a,b,err);
              if (Next_Sim_Val < Cur_Sim_Val) then
                begin
                  Cur_Sim_Val := Next_Sim_Val;
                  Cur_Col_Pos := a;
                  Cur_Row_Pos := b;
                  ClusterNum := 1;
                end
          end
    end; {for a}
  for b := 0 to arg-1 do            (* Search the second array *)
    for a := 0 to arg-1 do
      begin
        if (HMatReadEl(ptr2,a,b,err) < 1.0) then
          begin
            Next_Sim_Val := HMatReadEl (simptr2,a,b,err);
              if (Next_Sim_Val < Cur_Sim_Val) then
                begin
                  Cur_Sim_Val := Next_Sim_Val;
                  Cur_Col_Pos := a;
                  Cur_Row_Pos := b;
                  ClusterNum := 2;
                end
          end
    end; {for a}

(* The cluster corresponding to the array with the smallest distance  *)
(* value is incremented by one.  The pixel associated with the        *)
(* distance is turned on in Window Three in the appropriate color for *)
(* the cluster it is in.  The first cluster is the cloud cluster, and *)
(* will be displayed in the same colors as in the visible image.  The *)
(* second cluster is non-cloud, and will be displayed at a brightness *)
(* of 50.                                                             *)

if ClusterNum = 1 then
  begin
    HMatWrtEl (ptr2,Cur_Col_Pos, Cur_Row_Pos,1.0,err);
    Cluster_One_Size := Cluster_One_Size + 1;
    PutPixel(Cur_Col_Pos+xs,Cur_Row_Pos+ys,
             Parray[Cur_Col_Pos,Cur_Row_Pos,0]);
  end;

if ClusterNum = 2 then
  begin
    HMatWrtEl (ptr2,Cur_Col_Pos,Cur_Row_Pos,2.0,err);
    Cluster_Two_Size := Cluster_Two_Size + 1;
    PutPixel(Cur_Col_Pos+xs,Cur_Row_Pos+ys,50);
  end;

(* Display the iteration number currently underway in Window Four. *)

  Str(Iter, Counter);
```

```
  SetWindow(4);
  OutTextXY(35,130,'Iterations Completed =');
  SetColor(0);
  OutTextXY(218,130,#219);
  OutTextXY(225,130,#219);
  OutTextXY(232,130,#219);
  OutTextXY(239,130,#219);
  OutTextXY(246,130,#219);
  OutTextXY(253,130,#219);
  SetColor(255);
  OutTextXY(218,130,Counter);
  SetWindow(3);

(* Compute the average values for the new clusters. *)

if ClusterNum = 1 then
 begin
  Cluster_One_IR :=  Cluster_One_IR + Parray[Cur_Col_Pos,Cur_Row_Pos,1];
  Cluster_One_Vis := Cluster_One_Vis+ Parray[Cur_Col_Pos,Cur_Row_Pos,0];
  Cluster_One_X := Cluster_One_X + Cur_Col_Pos;
  Cluster_One_Y := Cluster_One_Y + Cur_Row_Pos;

  IR_Cluster_One_Val := Cluster_One_IR/Cluster_One_Size;
  Vis_Cluster_One_Val := Cluster_One_Vis/Cluster_One_Size;
  X_Cluster_One_Val := Cluster_One_X/Cluster_One_Size;
  Y_Cluster_One_Val := Cluster_One_Y/Cluster_One_Size;
 end;

if ClusterNum = 2 then
 begin
  Cluster_Two_IR :=  Cluster_Two_IR + Parray[Cur_Col_Pos,Cur_Row_Pos,1];
  Cluster_Two_Vis := Cluster_Two_Vis+ Parray[Cur_Col_Pos,Cur_Row_Pos,0];
  Cluster_Two_X := Cluster_Two_X + Cur_Col_Pos;
  Cluster_Two_Y := Cluster_Two_Y + Cur_Row_Pos;

  IR_Cluster_Two_Val := Cluster_Two_IR/Cluster_Two_Size;
  Vis_Cluster_Two_Val := Cluster_Two_Vis/Cluster_Two_Size;
  X_Cluster_Two_Val := Cluster_Two_X/Cluster_Two_Size;
  Y_Cluster_Two_Val := Cluster_Two_Y/Cluster_Two_Size;
 end;

(* Begin the loop to fill the similarity matrices with updated values *)

if ClusterNum = 1 then
 begin
 for b := 0 to arg-1 do
  for a := 0 to arg-1 do
   begin
    if (HMatReadEl(ptr2,a,b,err) < 0.0) then
     begin
      DiffX := XWeight * SQR(X_Cluster_One_Val -
               HMatReadEl(Xptr,a,b,err));
      DiffY := YWeight * SQR(Y_Cluster_One_Val -
               HMatReadEl(Yptr,a,b,err));
```

```
      VisPart := VisWeight * SQR(Vis_Cluster_One_Val -
                 HMatReadEl(Visptr,a,b,err));
      IRPart := IRWeight * SQR(IR_Cluster_One_Val -
                HMatReadEl(IRptr,a,b,err));
        TempResult := (VisPart+IRPart+DiffX+DiffY);
        SimVal := SQRT(TempResult);
        HMatWrtEl (simptr,a,b,SimVal,err);
      end
    end; {for r}
  end; {if}

(* Second similarity matrix *)

if ClusterNum = 2 then
 begin
 for b := 0 to arg-1 do
  for a := 0 to arg-1 do
   begin
    if (HMatReadEl(ptr2,a,b,err) < 0.0) then
     begin
      DiffX := XWeight * SQR(X_Cluster_Two_Val -
               HMatReadEl(Xptr,a,b,err));
      DiffY := YWeight * SQR(Y_Cluster_Two_Val -
               HMatReadEl(Yptr,a,b,err));
      VisPart := VisWeight*SQR(Vis_Cluster_Two_Val -
                 HMatReadEl(Visptr,a,b,err));
      IRPart := IRWeight * SQR(IR_Cluster_Two_Val -
                HMatReadEl(IRptr,a,b,err));
        TempResult := (VisPart+IRPart+DiffX+DiffY);
        SimVal := SQRT(TempResult);
        HMatWrtEl (simptr2,a,b,SimVal,err);
      end
    end; {for a}
  end; {if}

(* This routine lets the user get out of the looping process by       *)
(* pressing ESC on the keyboard.                                      *)

  if (keypressed = TRUE) then
    begin
      Choice := Readkey;
      if (Choice = ESC) then
        Goto interrupt3;
    end
end; {iter loop}            (* This is the end of the looping process *)

(* Housekeeping routines to shut down the clustering program.         *)
(* The appropriate routines will be executed based upon which option  *)
(* was selected from the main clustering menu.                        *)

interrupt3:
if (flag=0) and (flag2=0) then
  begin
```

```
    HArrFree(simptr);          (* Free up the memory space allocated    *)
    HArrFree(simptr2);         (* for the arrays.                       *)
    HArrFree(ptr2);
    HArrFree(Visptr);
    HArrFree(IRptr);
    HArrFree(Xptr);
    HArrFree(Yptr);
    SetWindow(1);
    Mark_Cluster_Box(xx[active],xy[active],arg); (* Remove box from   *)
    SetWindow(1);                                (* Window One.       *)
    Goto Finish;
  end;

if (flag=1) and (flag2=0) then
  begin
    HArrFree(simptr);          (* Free up the memory space allocated    *)
    HArrFree(simptr2);         (* for the arrays.                       *)
    HArrFree(ptr2);
    HArrFree(Visptr);
    HArrFree(IRptr);
    HArrFree(Xptr);
    HArrFree(Yptr);
    Goto Finish;
  end;

if (flag=0) and (flag2=1) then
  begin
    HArrFree(simptr);          (* Free up the memory space allocated    *)
    HArrFree(simptr2);         (* for the arrays.                       *)
    HArrFree(ptr2);
    HArrFree(Visptr);
    HArrFree(IRptr);
    HArrFree(Xptr);
    HArrFree(Yptr);
    SetWindow(1);
    Mark_Cluster_Box(xx[active],xy[active],arg); (* Remove box from   *)
    Goto Finish;                                 (* Window One.       *)
  end;

if (flag=1) and (flag2=1) then
  begin

    HArrFree(simptr);          (* Free up the memory space allocated    *)
    HArrFree(simptr2);         (* for the arrays.                       *)
    HArrFree(ptr2);
    HArrFree(Visptr);
    HArrFree(IRptr);
    HArrFree(Xptr);
    HArrFree(Yptr);
    Goto Finish;
  end;

ExitProg:
  SetWindow(1);
```

```
  Mark_Cluster_Box(xx[active],xy[active],arg); (* Remove box from     *)
  Goto Finish;                                 (* Window One.        *)

Finish:
  SetWindow(4);
  ClearViewPort;       (* Clear Window Four and redisplay the main    *)
  MoveCursor_Top;      (* TSIPS menu.                                 *)
  SetWindow(1);
  ShowHelpMenu;

end; {Procedure Cluster_Analysis}              (* End of program *)



(* This procedure is the first to execute when the clustering option  *)
(* is chosen from the main TSIPS menu.  This procedure allows the     *)
(* user to select the size of the clustering area he wishes to work   *)
(* with.                                                              *)

Procedure Pick_Cluster_Size;

LABEL

   interrupt;

VAR

   choice : char;
   xsize  : byte;
   begin
   DisplaySubMenu('Y', choice, 4);
   if (choice='1') or (choice='2') or (choice='3') or (choice='0') then
    begin
     case choice of
        '0'  : xsize :=  20;
        '1'  : xsize :=  50;
        '2'  : xsize := 100;
        '3'  : xsize := 150;
     end; {case}
    end
   else
    Goto interrupt;
   Cluster_Analysis(xsize); (* Proceed to the main clustering routine *)
   interrupt:
   ResetSubMenu(2);

end; {Procedure Pick_Cluster_Size}
```

Bibliography

1. Anderberg, Michael R. *Cluster Analysis For Applications.* New York: Academic Press, 1973.

2. Barnes, James C. and Michael D. Smallwood, *TIROS-N Series Direct Readout Services User's Guide.* Washington: U.S. Department of Commerce, NOAA/NESDIS, March 1982.

3. Brubaker, Thomas A. and others. "Ultrafast Algorithms for Cloud Data Analysis," *Digital Image Processing and Visual Communications Technologies in Meteorology,* edited by Paul Janota, Proc. SPIE *846*: 38-46 (1987).

4. Chance, Barbara A. and others. "Automated Meteorological Satellite Image Interpretation: An Aid to Short-Range Weather Forecasting," *Digital Image Processing and Visual Communications Technologies in Meteorology,* edited by Paul Janota, Proc. SPIE 846: 13-17 (1987).

5. Kaufman, Leonard, and Peter J. Rousseeuw, *Finding Groups in Data, An Introduction to Cluster Analysis.* New York: John Wiley & Sons, Inc., 1990.

6. Kelso, Thomas S., Instructor, Personal Correspondence. Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, 26 May 1991.

7. Larcomb, Capt Charles H. *Image Navigation of TIROS-N Weather Satellite Data.* MS Thesis, AFIT/GSO/ENS/89D-10. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1989 (A216 041).

8. Platbrood, Gérard, and Henri Barten. "Infrared Analysis of Low Temperature Ashed Coal Ashes and Their Classification by Application of Clustering Theory," *Analytical Chemistry, 57*: 2504-2510 (November 1985).

9. Tzu, Sun. *The Art of War.* Translation by Samuel B. Griffith. Foreword by B. H. Liddell Hart. New York: Oxford University Press, 1971.

10. Wannamaker, Brian, "An Evaluation of Digitized APT Data From the TIROS-N/NOAA-A, -J Series of Meteorological Satellites," *International Journal of Remote Sensing, 5:* 133-144 (1984).